VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Využití metod strojového učení pro testování kvality průmyslových výrobků

Using Machine Learning Methods for Automated Tests of Industry Products

Jakub Štefanský

Diplomová práce

Vedoucí práce: doc. Ing. Petr Bilík, Ph.D.

Ostrava, 2021

Abstrakt

Hlavním úkolem diplomové práce je rozbor strojového učení pro vizuální inspekce výrobků. Konkrétně vyhodnocení povrchových defektů na výrobcích. Oblast strojového učení zahrnuje širokou škálu algoritmů, ale pro diplomovou práci bylo rozhodnuto se zaměřit pouze na oblast hlubokého učení a konvolučních neuronových sítí.

Na začátku byla provedena rešerše oblasti inspekce povrchových defektů za využití neuronových sítí. Následoval rozbor nástrojů využitých pro vývoj neuronových sítí a s tím souvisejících knihoven. Poslední částí byla praktická realizace ve formě vytvoření modelu pro segmentační úlohu a natrénování na datasetu SMD součástek. V této práci jsou srovnány dva přístupy. První byl ve využití komerčního programu Cognex Deep Learning Studio a jeho nástrojů pro trénování modelů hlubokého učení. A druhý přístup byl v implementaci U-Net a SegDecNet architektur pomocí knihovny TensorFlow. V případě architektury U-Net byl navržený ještě doplňující test, spočívající v porovnání výsledků při trénování na celých snímcích anebo na snímcích, které byly rozděleny na menší části s definovaným překryvem. Program Cognex Deep Learning Studio byl mimo jiné využitý pro anotaci datasetu SMD součástek.

Výsledkem jsou evaluační metriky, které srovnávají model z komerční program Cognex Deep Learning Studio a implementovaných architektury U-Net a SegDecNet.

Klíčová slova

strojové učení, neuronové sítě, segmentace, LabVIEW, TensorFlow, defekt, inspekce vad, SMD, hluboké učení

Abstract

The main task of master thesis is the analysis of machine learning for visual inspection of products. Specifically, evaluation of surface defects on products. The field of machine learning includes a wide range of algorithms, but for the master thesis it was decided to focus only on the field of deep learning and convolutional neural networks.

At the beginning, a search was made of the area of surface defect inspection using neural networks. This was followed by an analysis of the tools used for the development of neural networks and related libraries. The last part was a practical implementation in the form of creating a model for a segmentation task and training on a dataset of SMD components. In this work, two approaches are compared. The first was the use of the commercial software Cognex Deep Learning Studio and its tools for training deep learning models. And the second approach was to implement U-Net and SegDecNet architectures using the TensorFlow library. In the case of the U-Net architecture, an additional test was proposed, consisting in comparing the results of training on whole images or on images that were divided into smaller parts with a defined overlap. The Cognex Deep Learning Studio was used to annotate the SMD component dataset.

The result is evaluation metrics that compare the model from the commercial Cognex Deep Learning Studio and the implemented U-Net and SegDecNet architectures.

Keywords

machine learning, neural networks, segmentation, LabVIEW, TensorFlow, defect, defect inspection, SMD, deep learning

Poděkování

Rád bych poděkoval všem, jejichž rady přispěly ke zpracování diplomové práce. Zvláště pak děkuji panu doc. Ing. Petr Bilík, Ph.D. za vedení mé diplomové práce.

Obsah

Seznam použitých symbolů a zkratek

CNN	Convolutional Neural Network
SMD	Surface Mount Device
AUC	Area Under Curve
ROC	Receiver Operating Characteristic
NN	Neural Nets
IoU	Intersectio over Union
RGB	Red Green Blue
BN	Batch Normalization

Seznam ilustrací

Seznam tabulek

Úvod

V průmyslové výrobě je rostoucím trendem automatizace výroby, která přináší i vyšší požadavky na robustní a přesnou detekci defektních výrobků. Pokud se jedná o komplexní vyhodnocení, tak klasické metody strojového vidění nemusí být dostačující, nebo můžou vést ke složitým algoritmů, u kterých bývá obtížné zajistit adaptaci na nový typ dat. Řešením těchto problému je využití metod strojového učení.

Diplomová práce je zaměřena na strojového učení v kontextu vyhodnocení vizuálních defektů na výrobcích. Konkrétně je zaměření na defekty strukturální charakteru (např. rýhy, nečistoty, praskliny apod.). Práce je rozdělena na několik dílčích cílů. Prvním je rozbor současného řešení vyhodnocení defektů na výrobcích s využitím technik strojového učení. Druhým krokem je úvodní teoretický rozbor do oblasti strojového učení a jeho technik, následované bližším rozborem metod hlubokého učení. Oblast strojového učení je obsáhlá a jednou z podoblastí je hluboké učení, na kterou je tato diplomová práce zaměřena. Třetím krokem je popis využitých nástrojů pro implementaci metod hluboké učení a souvisejících knihoven. Předposledním krokem je anotace datasetu snímků, které jsou použity pro trénování modelů. Poslední části je implementace metod hlubokého učení pomocí popsaných nástrojů a vytvoření aplikace pro vyhodnocení kvality predikcí na anotovaném testovacím datasetu snímků.

Pro diplomovou práci byl zvolen dataset SMD součástek. SMD součástky mají širokou variaci defektů, kde hlavní úkolem je rozdělit snímky na ty, co jsou v pořádku - OK a ty, co mají vadu - NOK. Pro vyhodnocení datasetu SMD součástek byly zvoleny dva hlavní přístupy. První je využití komerčního nástroje Cognex Deep Learnig Studio, který je přímo určený pro vyhodnocení kvality průmyslových výrobků. Druhý přístup spočívá ve využití open source architektur neuronových sítí za využití knihovny TensorFlow. Ve druhém přístupu bylo zvoleno a implementováno několik rozdílných technik a architektur pro srovnání odlišných pojetí na datasetu SMD součástek.

V první kapitole je úvod do strojového učení a jeho metod. Dále jsou blíže specifikovány cíle diplomové práce, rešerše odborných publikací souvisejících s problematiku strojového učení v oblasti vyhodnocení defektů na výrobcích.

Ve druhé kapitole je teoretický rozbor oblasti hlubokého učení a neuronových sítí. V kapitole je zaměření na konvoluční sítě a dalších operací související s problematikou segmentace obrazu.

Ve čtvrté kapitole je provedený rozbor použitých nástrojů. Použitými nástroji je komerční program Cognex Deep Learnig Studio, open source knihovna TensorFlow a Keras. Dále je popsána knihovna Albumentations související s tvorbou augmentačních sekvencí. A nakonec vývojové prostředí LabVIEW, u kterého jsou popsané hlavní knihovny použité pro tvorbu testovací aplikace.

V páté kapitole je v jednotlivých podkapitolách rozvedený postup praktické realizace. První je rozbor použitého datasetu SMD součástek. Druhou částí je rozbor evaluačních metrik na základě, kterých se hodnotí kvalita predikce modelů na použitém datasetu. Následuje podkapitola zabývající se prací v programu Cognex Deep Learnig Studio. Předposlední částí jsou popsaný architektury a metody využité pro trénovaní modelů s knihovnou TensorFlow a Keras. Poslední částí praktické realizace je popis aplikace pro testovaní natrénovaných neuronových sítí vytvořený v LabVIEW.

Poslední kapitola představuje srovnání výsledků z Cognex Deep Learnig Studia a architektur implementovaných v knihovně TensorFlow a Keras.

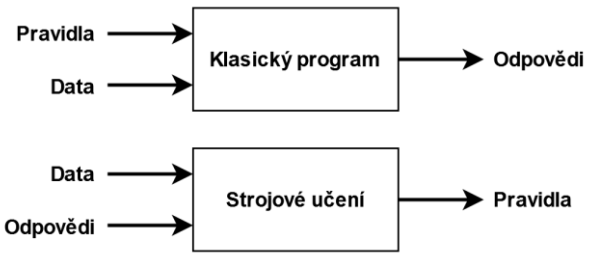
1 Úvod do problematiky strojového učení

Cílem diplomové práce vycházející ze zadání je rozbor problematiky strojového učení pro vizuální inspekce výrobků s cílem detekce defektů. Tento cíl lze rozdělit na několik dílčích částí. První je rešerše dostupných publikací zabývající se problematikou vizuální inspekce výrobků s využitím technik strojového učení. Druhá část představuje teoretický rozbor strojového učení a následované rozborem dílčí oblasti strojového učení, která je využitá pro praktickou realizaci. Třetí část je vybrání a anotování datasetu snímků, na kterých se provede trénování vybraných technik strojového učení. Vybraný dataset musí obsahovat snímky výrobků, které obsahující defekty strukturálního charakteru. Poslední částí je vytvoření aplikace, která provede samotné vyhodnocení natrénovaných modelů na testovacím datasetu snímků.

V podkapitole 1.1 je popsané strojové učení a jeho podoblasti. V podkapitole 1.2 je rozvedené současné řešení týkající se vizuální inspekce výrobků.

1.1 Strojové učení

Strojové učení je počítačová věda, která usiluje o učení počítače na dodaných datech. Cílem oboru je dát schopnost počítačům se učit bez nutnosti explicitního naprogramování. Na Obr. 1.1 je ukázaný rozdíl mezi klasickým programováním a strojovým učením. V případě klasického programování vytvoří člověk program, který definuje pravidla. Na základě pravidel definovaných v programu se provede rozdělení dat a výstupem jsou odpovědi. V případě strojového učení se dodají data spolu s odpověďmi (anotace) očekávanými při zpracování dat. Výsledek zpracování jsou pravidla, která lze potom využít na nová data. [4] [10]



Obr. 1.1 *Strojové učení jako nové programové paradigma [4]*

Systém s algoritmy strojového učení je natrénován, a to způsobem předložení dostatečného množství relevantních dat. V předložených datech nalezne statistickou strukturu, ze kterých odvodí pravidla. Narozdíl od klasické statistiky má strojové učení schopnost se vypořádat s rozsáhlými datovými množinami a komplexními vlastnostmi dat (např. obrazová a zvuková data). [4]

Strojové učení je vhodné pro případy [10] :

1. Problém, ve kterém existující řešení vyžaduje značné množství manuálního ladění, anebo vytváření dlouhého listu pravidle. Algoritmy strojové učení můžou výrazně zjednodušit zpracování daného problému.
2. Komplexní problém, pro který není vhodný řešení ve všech oblastech tradičních přístupů (např. klasické zpracování obrazu)

3. Proměnlivé prostředí. Systém strojového učení se může přizpůsobit novým datům.
4. Zpracování velkého množství dat, anebo komplexní data.

Před zahájením učení je potřeba definovat tři věci: vstupní data (např. obrazová data), cílové hodnoty (nebo také anotace, např. třídy OK a NOK apod.) a nakonec způsob měření, který vyhodnotí, jestli se algoritmus správně učí. Měření je definované jako zpětnovazební signál pro úpravu parametrů algoritmu, toto nastavení je označované jako proces učení. Model strojového učení převádí vstupní dat na požadovaný výstup, a proto je nejdůležitějším prvkem modelů je smysluplná transformace dat a tím pádem schopnost se naučit reprezentace dodaných dat. Pod pojmem reprezentace je možné si představit zakódovaná data, příkladem můžou být barevné formáty RGB a HSV, které zakódují stejná data, ale přestavují dvě rozdílné reprezentace dat. [4]

1.1.1 Dělení metod strojového učení

Strojové učení je obsáhlý obor s celou řadou podoblastí. Algoritmy je možné dělit podle různých parametrů, ale základním rozdělení se provádí podle zapojení lidského dohledu, a to do celkem čtyř základních kategorií: řízené, neřízené, samořízené a posilované učení.

Řízení učení (supervised learning), nebo také učení s učitelem je nejběžnější podoblastí strojového učení. Učení je založené na mapování vstupních dat na známé cíle (anotace). Nejdůležitějšími algoritmy v podoblasti řízeného učení jsou [10]:

- Rozhodovací stromy (Decision Trees) a náhodné lesy (Random Forests)
- Lineární regrese
- Support Vector Machines (SVM)
- K- nejbližších sousedů (k-Nearest Neighbors)
- Neuronové sítě

Neřízené učení (unsupervised learning), nebo také učení bez učitele, spočívá v hledání transformací dat bez dodání cílových hodnot. Účelem transformací může být odstranění šumu nebo nalezení korelací ve vstupních datech. Nejdůležitějšími kategoriemi v podoblasti neřízeného učení jsou [4][10]:

- Shlukování (Clustering): k-Průměry (k-Means), hierarchická shluková analýza (Hierarchical Cluster Analysis - HCA)
- Vizualizace a redukce dimenze: Principal Component Analysis (PCA)

Redukce dimenze je technika, jejímž cílem je zjednodušení dat s malou ztrátou informací. Jedním ze způsobů je sloučení několika atributů s vysokou korelací do jediného atributu. Další z technik neřízeného učení je detekce anomálií, kde systém je učený na normálních instancích. Při vyhodnocení na nových datech může říct, zda se jedná o normální instanci, nebo se jedná o anomálii. [4][10]

Samořízené učení (self-supervised learning) je speciálním typem řízeného učení. Jedná se o řízené učení bez anotací vytvořených člověkem. Anotace jsou automaticky generovány ze vstupních dat (obvykle pomocí heuristického algoritmu). Jedním z příkladů může být autoenkodér, kde cílem generovaného výstupu jsou nemodifikovaná vstupní data. [4]

Posledním je posilované učení (reinforcement learning), kde učící systém je označený jako agent, který může pozorovat prostředí, zvolit a vykonat akci. Na základě provedené akce je agent odměněn, nebo naopak penalizovaný. [10]

Na základě popsaného úvodu do strojového učení a jeho metod je zřejmé, že se jedná o obsáhlý obor. A proto bylo rozhodnuto se v této diplomové práci zaměřit pouze na oblast neuronových sítí s řízeným učením, které jsou dominantní ve strojovém učení.

1.2 Rešerše publikací

Oblast hlubokého učení je v současnosti jedna z nejrychleji se rozvíjejících odvětví počítačových věd, a to z důvodu schopnosti řešit vysoce komplexní problémy. Akumulací tradičních technik strojového učení vyústila v evoluci hlubokého učení, které také získalo inspiraci ze statistického učení. V tradičním strojovém učení je potřeba provést tzv. konstrukci příznaků (feature engineering). Je to proces, při kterém je potřeba zvolit správné transformace pro získání potřebných příznaků. Nicméně u technik hlubokého učení se provede naučení komplexních a abstraktních reprezentací, které umožní transformaci dat pro získání potřebných příznaků (feature learning). [4]

Rešerše je zaměřena na publikace, které se zabývají oblastí neuronových sítí. Vytvořená rešerše představuje průřez zásadními odbornými publikacemi týkající se inspekce průmyslových výrobků, a to se zaměřením povrchové defekty.

Publikace je možné rozdělit podle typu úlohy, na kterou se zaměřují. Základními typy úloh jsou klasifikace, lokalizace a segmentace. Cílem klasifikační úlohy je klasifikovat snímky do definovaných tříd. Druhý typ úlohy je lokalizace, která přestavuje hledání objektu ve snímků. Pro každý nalezený objekt se vygeneruje ohraničující rámeček, který specifikuje přibližnou polohu hledaného objektu ve snímku. Poslední typ úlohy segmentace provádí hledání objektu v obraze, ale na rozdíl od lokalizace, pro nalezení objektu na pixelové úrovni. Výstupem ze segmentace je mapa pixelů definující nálezné třídy objektů. Ještě jinak řečeno u segmentační úlohy se vyhodnotí každý pixel vstupního snímku, výstupem je mapa pixelů, kterým přiřazena třída.

Publikace Weimer et al. [11] z roku 2016 pracuje s datasetem DAGM 2007. Jedná se o dataset uměle vytvořený pro účely testování detekce defektů. Dataset má celkem šest tříd, v každé třídě je 1000 snímků bez defektů a 150 snímků s defekty. Snímky mají rozlišení 512x512. V případě publikace se využívá klasická konvoluční neuronová síť (dále jen CNN – Convolution Neural Network) a je zakončená klasifikátorem s plně propojenými vrstvami. Výsledkem byla v klasifikaci snímků, při které dosáhli přesnosti 97,03 %. Další publikace Yu-ting LIU et al. [12] pracuje také s datasetem DAGM 2007, ale provádí segmentaci defektů, a to pomocí architektury FCN. S architekturou FCN se jim podařilo dosáhnout přesnosti 99,6 %, ale využili pouze čtyři třídy z celkových šesti a zároveň omezili celkový počet snímku z datasetu DAGM 2007, a proto není možné srovnat s předešlou publikací Weimer et al. [11].

Další z publikací je Hui Lin et al. [13] provádí detekci defektů na LED čipech. Využili dva datasety s rozdílnými pouzdry čipů. První z datasetů obsahuje 24000 snímků a z toho 8000 je bez defektů. Druhý dataset má 10400 snímků a z toho 2400 je bez defektů. Pro zpracování využili klasickou CNN s klasifikátorem. Architektura je označena jako LEDNet. Kromě klasifikace ještě vytvořili tzv. CAM (Class Activation Mapping), který se provede sloučením filtrů z poslední konvoluční vrstvy a tím se získá

predikční mapa. Vzniklou mapu využily pro vygenerování ohraničujícího rámečku, který přibližně lokalizoval polohu defektu. Celkem se jim podařilo dosáhnout u prvního datasetu nepřesnosti 5,04 % a u druhého datasetu nepřesnost byla 5,51 %.

Publikace Zhenshen Qu et al. [14] prezentuje architekturu PartsNet určenou pro detekci defektů na dílech z motoru. Architektura je založena na CNN realizující segmentační úlohu a dále obsahuje síť určenou pro vylepšení příznaků. CNN slouží k nalezení základních oblastí s defekty na výrobcích. Poté se několik tradičních metod, které se používají ke zpřesnění výsledků segmentace, transformuje na konvoluční vrstvu. Tyto metody se sestaví do sítě s pevnými váhami a empirickými prahovými hodnotami. Tyto prahové hodnoty jsou poté povoleny, aby se zlepšila jejich adaptační schopnost a realizoval se trénink celé sítě. Dataset obsahuje celkem 5 typů defektních výrobků, pro každý typ je dostupných 500 snímků a pro testování 100 snímků s defekty a 100 snímků bez defektů. Výsledky překryvu nalezených defektů a anotací dosahuje kolem 90 %.

V publikaci Domen Tabernik et al. [7] navrhli architekturu CNN, která kombinuje úlohu binární klasifikace a binární segmentace. Na výstupu ze sítě je poskytnutá predikční mapa a klasifikační skóre určující, jestli se jedná defektní výrobek. Trénování je prováděné nejdříve trénováním segmentační části sítě a klasifikační je zmražená (není umožněné trénování). Po natrénování segmentační části se provede zmražení této části, a naopak se trénuje klasifikační část. Na vyhodnocení byl využitý dataset KolektorSDD se snímky z elektrických komutátorů, na kterých se hledají se praskliny. Velikost snímků je 1408x512. Výsledné srovnání zahrnovalo architektury U-Net, DeepLab v3+ a nakonec komerční software Cognex ViDi Suit. Nejlepších výsledků v klasifikaci snímků bylo dosažené s jejich vlastní architekturou SegDecNet a hned následované s výsledky z Cognex ViDi Suit. Zbylé dvě dosahovali horších výsledků, a to hlavně v počtu falešných predikcí a neodchycených defektů. Nevýhoda jejich architektury SegDecNet je, že poskytuje predikční mapy, které nejsou ve stejné velikosti jako jsou vstupní snímky, ale jsou výrazně zmenšené. Tento způsob je udržitelný pro větší snímky, ale není vhodný malé snímky, protože by došlo ke ztrátě drobných defektů vlivem nízkého rozlišení.

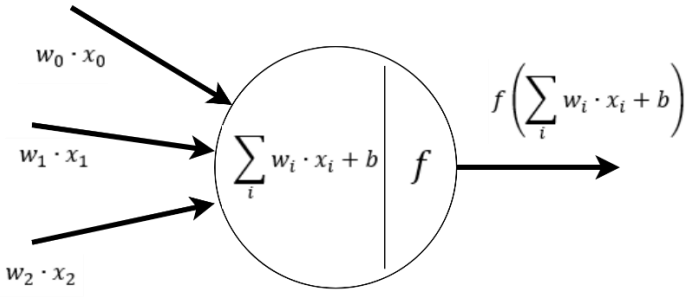
Publikace Yang Liu et al. [15] provádí hledání defektů na ocelovém plechu válcovaném za tepla. Jejich přístup je založený na klasické CNN a na long short-term memory (dále jen LSTM). Architektura je navržena pro detekci periodických defektů, jako jsou značky rolování. Nejprve jsou defektní části snímku extrahovány prostřednictvím sítě CNN a poté jsou extrahované příznaky vloženy do sítě LSTM pro rozpoznání defektu. Další z publikací je Xian Tao et al. [16], která provádí detekci poškození na kovovém povrchu, kde se provádí detekce prachu, rýh apod. Využívají k tomu vlastní navrženou architekturu CASAE, které implementuje kaskádu autoenkoderů pro extrakci predikčních map. Na základě predikčních map se provede vyříznutí nalezených oblastí ze vstupního snímku. Vyříznuté oblasti se následně vyhodnotí v klasické CNN s klasifikátorem, který predikuje typ defektu.

2 Neuronové sítě

Označované také jako umělé neuronové sítě (artificial neural networks - ANNs). Neuronové sítě jsou základem tzv. hlubokého učení (deep learning). Oblast hlubokého učení a neuronových sítí je podoblastí strojového učení. V následujících odstavcích je blíže rozvedený úvod do neuronových sítí.

Hluboké dopředné sítě, také označované dopředné(feedforward) neuronové sítě, nebo vícevrstvé preceptronové sítě (zkratka MLP), jsou základním prvkem modelů hlubokého učení. Cílem dopředných sítí je aproximovat nějakou funkci f^*. Například pro klasifikátor $y = f^*(x)$ mapujeme vstup x na třídu y. Dopředné neuronové sítě konkrétně definují mapování jako $y = f(x; \theta)$, kde θ jsou parametry neuronové sítě, které se snažíme naučit, tak aby co nejlépe aproximovaly funkci.[18]

Dopředné neuronové sítě se skládají ze základních bloků, které jsou označované jako preceptron. Na Obr. 2.1 je ukázka preceptronu. Vstupem jsou výstupy z předešlých preceptronů x_i, které jsou vynásobeny váhami w_i. Váhy w_i představují parametry, které se snažíme učením nastavit, tak aby prováděli potřebnou transformaci. Následně se provede sečtení těchto vstupních hodnot, a nakonec se vloží do tzv. aktivační funkce f. Aktivační funkcí může být např. funkce ReLU, Sigmoid apod. Pokud by se nepoužila aktivační funkce, tak daný preceptron by se mohl naučit pouze lineární transformaci vstupních dat. Lineární transformace, ale nejsou dostačující pro složitější a komplexnější úlohy. Proto se využívá právě aktivační funkce, která vkládá nelinearitu do neuronových sítí. [4][20]



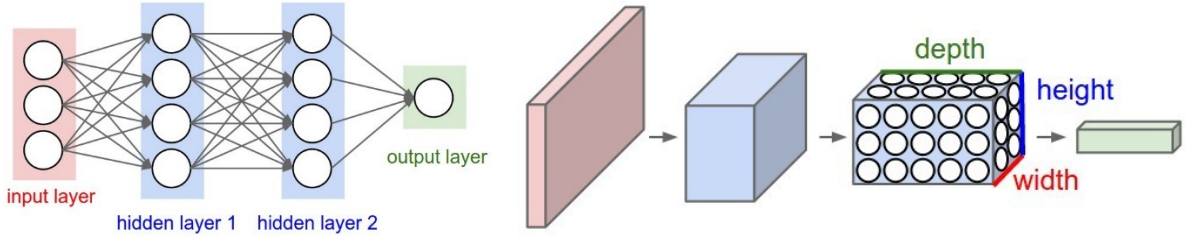
Obr. 2.1 *Preceptron [20]*

Samotné učení neuronových sítí probíhá tak, že se na vstup neuronové sítě přivedou data, na kterých se daná síť spustí a poskytne na základě vstupu nějakou predikci na výstup. Predikce se použije pro výpočet tzv. ztrátové funkce, kde jejími vstupními hodnotami jsou predikce a cílové hodnoty (anotace). Potom se spočítá gradient ztráty s ohledem na parametry sítě. A nakonec se provede přesun parametrů opačným směrem od gradientu, proto abychom našli ideálně lokální minimum aproximující funkci f^*. Tento postup učení je označený jako stochastický gradientní sestup. Neuronová síť se skládá z mnoha operací, které jsou spojené dohromady. Každá z těchto funkcí má známou derivaci. Abychom mohli spočítat jednotlivé složené derivace pro získání gradientu, tak je k tomu potřeba využít tzv. algoritmus zpětného šíření (backpropagation). Algoritmus začíná na výstupu z neuronové sítě s hodnotou ztráty. Postupně se posouvá od horních vrstev(výstupní) až po spodní vrstvy(vstupní). Pro výpočet příspěvku každého parametru ke ztrátě se využije derivace složené funkce. [4]

2.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou obdobou běžných neuronových sítí (plně propojených sítí). Jedná se o sítě využívané pro práci s obrazovými daty. Hlavním rozdílem je, že plně propojené sítě se učí globální vzory. Naproti tomu konvoluční sítě se učí lokální vzory. Tato vlastnost poskytuje dvě velké výhody. První výhodou je, že vzory, které se učí jsou translacím invariantní. Například, když se konvoluční síť naučí defekt v horním rohu snímku, tak potom zvládne rozpoznat stejný defekt v dolním rohu snímku, přestože nebyla naučená na defekt, který je v dolním rohu. Další výhoda je, že se učí prostorové hierarchie vzorů, tato vlastnost umožňuje vytvářet konvoluční neuronové síti komplexní a abstraktní vizuální obrazce. [4]

Na Obr. 2.2 je srovnání klasické neuronové sítě – vícevrstvá preceptronová síť a napravo je konvoluční neuronová síť (dále jen CNN). Na pravém obrázku může považovat červený blok jako vstupní obraz, který má šířku, výšku a hloubku. Hloubka je v případě RGB obrazu 3 a obraz ve stupních šedi má hloubku 1. Tento vstupní obraz je následně transformovaný pomocí 2D konvoluční vrstvy do 3D tenzoru. [20]



Obr. 2.2 *Srovnání klasické neuronové sítě(nalevo) a CNN(napravo)[20]*

Ve vzorci (2.1) [18] je ukázaný výpočet diskrétní 2D konvoluce, kde K představuje tzv. jádro (kernel) a I je např. vstupní obraz na kterém se provádí konvoluce. Jádro se nejčastěji nastavuje velikost 3x3, nebo 5x5. Další důležitý parametr v případě konvoluční vrstvy je hloubka mapy výstupních funkcí, což představuje počet filtrů, který se vypočítá konvolucí. Často se začíná počtem filtrů 32 a postupně se zdvojnásobuje. [4]

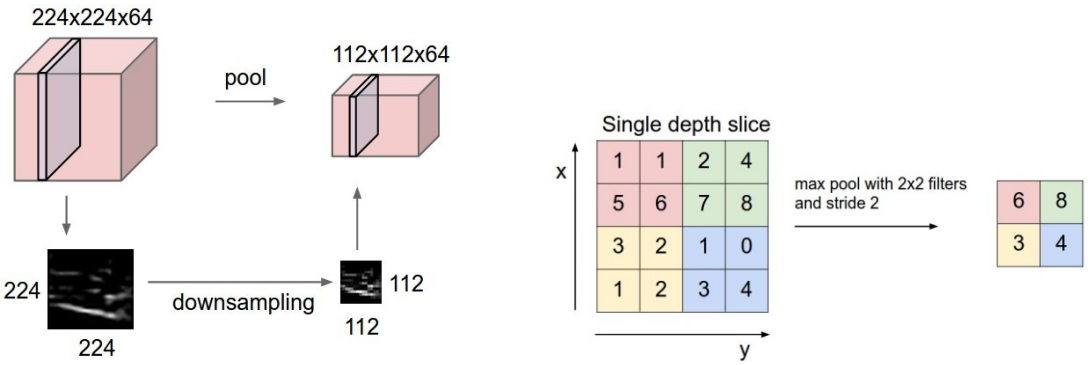
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \tag{2.1}$$

2.2 Další operace v neuronových sítí

2.2.1 Vrstva sdružovaní dle maxima

Je běžnou praktikou periodické vkládání vrstvy sdružování dle maxima (Max-pooling) při tvorbě architektur konvolučních neuronových sítí. Vrstva se vkládá za konvoluční vrstvu. Jejím účelem je postupná redukce parametrů neuronové sítě. Provádí převzorkování dat, a tudíž neobsahuje žádné parametry pro trénování. Tím se sníží výpočetní náročnost a zároveň se sníží šance na přetrénování sítě. Parametry vrstvy jsou velikost filtru a krok. Velikost filtru určuje, jak velkou oblast má funkce zahrnout pro získání maximální hodnoty. Krokem je určená míra převzorkování. Nejčastěji se nastavuje filtr s velikostí 2x2 a krokem 2. Na Obr. 2.3 je ukázka procesu převzorkování mapy příznaků s filtrem 2x2 a krokem 2 – vrstva sdružení dle maxima. Existuje také vrstva sdružení dle průměru (Average-

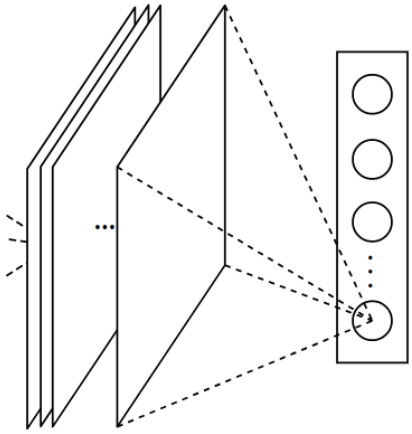
pooling). Tato vrstva se příliš nevyužívá, protože je mnohem zásadnější zjistit maximální přítomnost příznaků než jejich průměrnou přítomnost. [4][20]



Obr. 2.3 *Vrstva sdružení dle maxima [20]*

2.2.2 Vrstva globálního sdružení

Konvenční CNN pro klasifikaci využívají plně propojené vrstvy, které na základě příznaků z konvoluční části provedou klasifikaci na definované třídy. Nevýhoda tohoto řešení je, že plně propojené vrstvy představují podstatnou část trénovacích parametrů. Plně propojené vrstvy můžou výrazně zvýšit počet parametrů, a to potom může vést k přetrénování a tím dojde ke snížení generalizační schopností neuronové sítě. Z těchto důvodů se využívá tzv. drop out (výpadek), který může omezit přetrénování, a to způsobem náhodného odhození některých trénovacích parametrů. Další možností je využít vrstvu globálního průměrného anebo maximálního sdružení. Tato vrstva nahradí plně propojené vrstvy. Výhodou je, že neobsahuje žádné trénovací parametry a může tedy pomoci předejít přetrénování neuronové sítě. [19]



Obr. 2.4 *Globální sdružení [19]*

2.2.3 Vrstva dávková normalizace

V rámci trénování neuronových sítí je správným postupem normalizace dat. První částí normalizace jsou vstupní dat do neuronové sítě. Například obrazová data mají hodnotu pixelů [0;255], ale počáteční hodnoty neuronové sítě se pohybují v rozsahu [0;1]. Tímto velkým rozdílem hodnot může dojít k vypnutí rozsáhlých aktualizací gradientu a síť nebude konvergovat. Z těchto důvodů je potřeba

zajistit, aby data byly homogenní, a tedy normalizovat vstupní data na rozsah [0;1]. Zároveň existuje vrstva dávkové normalizace (Batch Normalization), která provádí reparametrizace dané neuronové sítě, ve které je implementovaná. Vrstva se zpravidla aplikuje za konvoluční vrstvou. Výhodou použití dávkové normalizace je zlepšení průchodu gradientu během trénování. Ve vzorci (2.2) je ukázka, jak probíhá normalizace. Kde H představuje výstup např. z konvoluční sítě a H' je normalizovaná dávka trénovaných dat. Pomocí vzorců (2.3) a (2.4) se následně spočítá průměr a standartní odchylka. [18][4]

$$H' = \frac{H - \mu}{\sigma} \tag{2.2}$$

$$\mu = \frac{1}{m} \cdot \sum_i H_i \tag{2.3}$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \cdot \sum_i (H - \mu)_i^{\,2}} \tag{2.4}$$

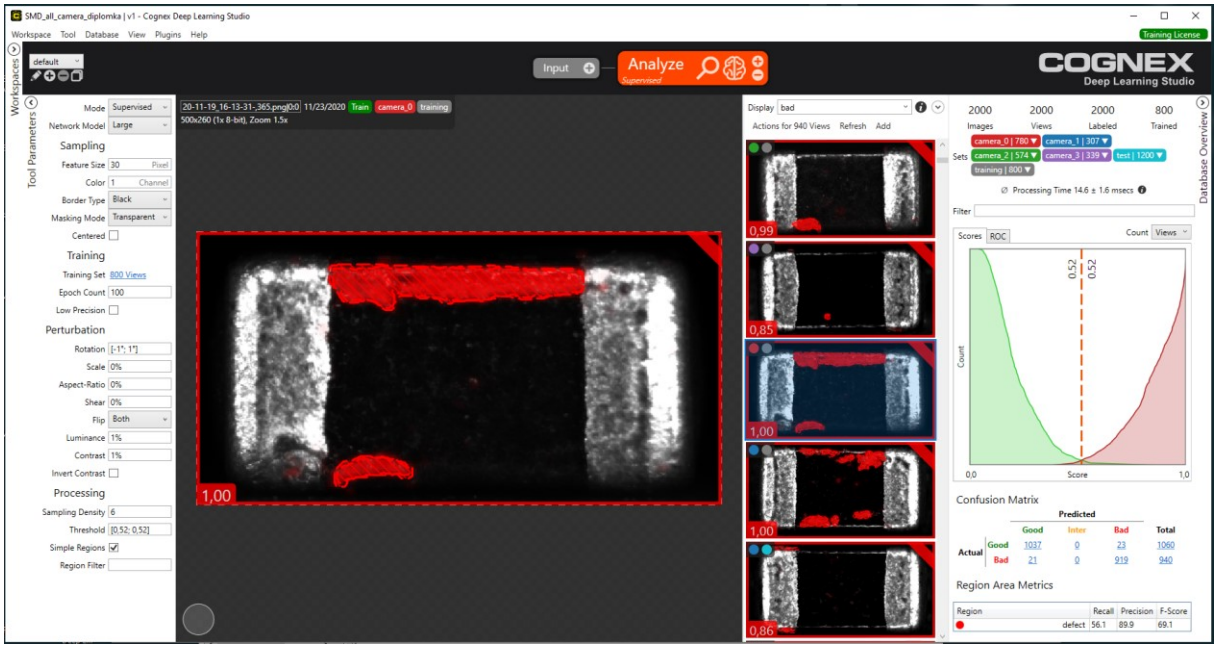
3 Rozbor nástrojů

Pro vývoj neuronových sítí je možné přistupovat několika způsoby. První možností je využít dostupných open source frameworků pro vývoj neuronových sítí. Mezi nejpoužívanější patří TensorFlow, PyTorch, Apache MXNet, CNTK a Caffe2. Druhá možnost je využití komerčních produktů založených na trénování neuronových sítí. Pro praktickou část diplomové práce byly zvoleny oba přístupy. Konkrétně byl zvolený pro natrénovaní modelu komerční produkt Cognex VisionPro Deep Learning, který byl mimo jiné využit pro anotaci obrazových dat. Druhý nástroj byl vybraný framework TensorFlow s jazykem Python.

V následujících podkapitolách jsou popsány oba zvolené nástroje, spolu s další dodatečnými nástroji využitými při vývoji.

3.1 Cognex Deep Learning Studio

Cognex VisionPro Deep Learning je komerční software, poskytující soubor nástrojů pro strojové vidění, které využívají techniky hlubokého učení. Pro práci s nástroji je určený program s grafickým rozhraním Cognex Deep Learning Studio (dále zkráceně Cognex DLS), který je učený pro trénování nástrojů a anotaci obrazových dat. Na Obr. 3.1 je ukázka prostředí Cognex DLS s otevřeným projektem a nástrojem Red Analyze.



Obr. 3.1 *Ukázka Cognex Deep Learning Studio – nástroj Red Analyze*

Cognex DLS poskytuje volba celkem ze čtyř nástrojů: Green Classify, Blue Locate, Blue Read a Red Analyze.

První nástroj je Green Classify, který je učený pro klasifikaci snímků. Green Classify nabízí volbu mezi dvěma režimy Focused a High Detail. Režim Focused se používá ke klasifikaci objektu nebo celé scény. Ať už je to identifikace výrobků na základě jejich obalu, klasifikace svarů nebo oddělení přijatelných nebo nepřijatelných vad. Režim Focused se naučí oddělit různé třídy na základě kolekce

označených obrázků. Druhý režim High Detail nabízí jinou architekturu, určenou pro složitější úlohy, kde je nutné zajistit nejvyšší přesnost. Druhý režim poskytuje i vizuální zpětnou vazbu v podobě tzv. heat mapy, která vizualizuje oblast, na kterou se daný nástroj zaměřuje.

Další nástroj Blue Locate se používá k vyhledání a lokalizaci jedné nebo více objektů v obraze, od malých objektů na zašuměném pozadí až po složité objekty hromadně. Nástroj Blue Locate dokáže lokalizovat a identifikovat složité prvky a objekty učením se z anotovaných obrázků.

Předposlední nástroj Blue Read se používá k provádění optického rozpoznávání znaků (OCR) v obraze. Nástroj Blue Read identifikovat a číst znaky učením se z anotovaných obrázků. Čtení dovode provádět z čistě vytištěných znaků až po silně deformované znaky na zašuměném pozadí.

Poslední nástroj Red Analyze je v následující podkapitole 3.1.1 popsaný podrobněji, protože je to hlavní nástroj, který je využitý v této diplomové práci.

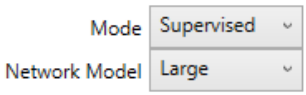
3.1.1 Nástroj Red Analyze

Nástroj Red Analyze je možné zvolit ve dvou režimech: Unsupervised (bez učitele) a Supervised (s učitelem). Režim Unsupervised se používá k detekci anomálií a estetických vad. Ať už jsou to škrábance, neúplné nebo nesprávné sestavy nebo poškození textilu. V tomto režimu je nutné poskytnou pouze dobré kusy (bez poškození) a na základě těchto snímku se naučí nástroj. Druhý režim Supervised se používá k segmentaci konkrétních oblastí, jako jsou defekty nebo jiné oblasti zájmu. Nástroj Red Analyze v režimu s učitelem dokáže identifikovat defekty a to tak, že se naučí různý vzhled defektu nebo cílové oblasti. Pro natrénovaní je nutné poskytnout snímky s anotovanými vadami.

Cognex DLS umožňuje anotaci snímků. Na Obr. 3.2 je ukázka všech dostupných nástrojů učených pro anotaci defektů na snímcích. První nástroj zleva je Line, který je určený pro obkreslení kontur defektu. Parametr Width určuje šířku kreslící nástroje Line. Je dostupný i nástroj Erase, který je určený pro smazání nakreslených anotací. Po dokončení kreslení anotací je možné uložit anotaci pomocí tlačítka Apply.

Obr. 3.2 *Nástroje na anotaci*

Před samotným trénováním na anotovaných datech, je nutné nastavit několik zásadních parametrů. První je nastavení modelu (Obr. 3.3), kde je možné zvolit režim Supervised (s učitelem) a Unsupervised (bez učitele). A nakonec je na výběr velikost modelu ve variantách: Small, Medium a Large.

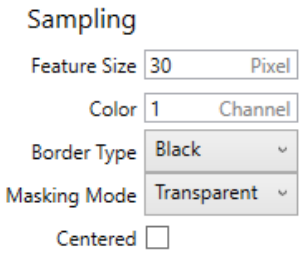


Obr. 3.3 *Nastavení modelu*

Dalším krokem je nastavení vzorkování snímku a další souvisejících parametrů (Obr. 3.4). První je Feature size, kterým se nastavuje v pixelech velikost hledaných defektů. Je to parametr, který v podstatě definuje velikost vzorku. Ve skutečnosti nástroj Red Analyze při učení vidí 5x větší plochu, než je definovaná Feature size, ale při vyhodnocení je důležitost kladena směrem ke středu vzorku. Feature

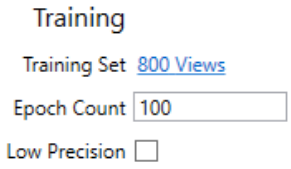
size má závislost na rychlost vyhodnocení modelu, například Feature size 100 je 100x rychlejší než Feature size 10.

Další parametr (Obr. 3.4) je nastavení barevného kanálu snímku. Při vzorkování dochází vyextrahování menších bloků snímků, pokud se provádí na okrajích snímků, tak jsou vzorky neúplné a je nutné tuto oblast doplnit buď nulami, nebo provést replikaci. Toto doplnění se nastavuje parametrem Border Type. Poslední dva parametry se týkají maskovaní neužitečných, nebo nežádoucích parametrů, ale vzhledem k tomu, že není využité maskovaní, tak nebudou blíže popsány.



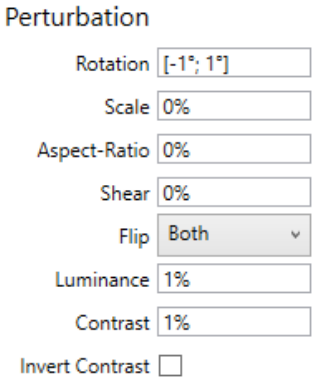
Obr. 3.4 *Nastavení vzorkování snímků*

Pro možnost trénování je nutné nastavit na jakých snímcích se bude trénovat. Volbu je možné udělat přes parametr Traning Set (Obr. 3.5). Důležitý parametrem během trénování je počet epoch (Epoch Count), po které se má model trénovat. Poslední je nastavení Low Precision, který výrazně zvýší rychlost vyhodnocení modelu, ale za cenu nižší přesnosti.



Obr. 3.5 *Nastavení trénování*

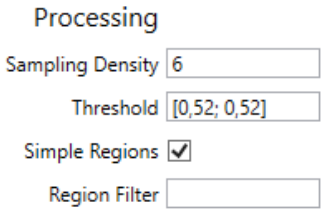
V rámci trénování modelů je běžné nastavovaní augmentace dat. V Cognex DLS je možné pomocí sady parametrů Perturbation (Obr. 3.6). Je dostupné nastavení: rotace, škálování, úprava poměru stran, zkosení, překlopení, jas, kontrast a invertování kontrastu.



Obr. 3.6 *Nastavení augmentace snímků*

Poslední skupina parametrů se týká vyhodnocení modelu. První je Sampling Density, který nastaví hustotu vzorkovaní vzhledem k Feature size.

Model poskytuje na výstupu klasifikační skóre snímku od 0 do 1, které udává pravděpodobnost, že se jedná o defektní snímek. Proto je nutné nastavit práh (parametr Threshold) definující, kdy se jedná o defektní výrobek. Parametr Simple Regions nastaví, že nalezené defekty musí být celistvé (bez děr). Poslední parametrem je možné definovat minimální velikost defektu.



Obr. 3.7 *Nastavení vyhodnocení modelu*

Po nastavení všech parametrů a anotování snímků, je možné zahájit trénování modelu. Po dokončení trénování modelu se automaticky provede evaluace celého datasetu snímků. Pokud model poskytuje dostatečné výsledky, tak je možné vyexportovat model, který Cognex označuje jako tzv. Runtime Workspace ve formátu VRWS.

VisionPro Deep Learning poskytuje integraci jejich runtime modelu pomocí knihoven jazyka C anebo knihovny pro .NET. Pro využití .NET knihoven je možné je doinstalovat do Visual Studia pomocí NuGet packages, které se nachází v podadresáři aplikace Cognex DLS.

Celá podkapitola 3.1 byla čerpaná z VisionPro Deep Learning Help [5].

3.2 TensorFlow

TensorFlow je open source softwarová knihovna pro strojové učení vytvořena společností Google, která umožňuje vytváření a vyhodnocení data flow grafů pro numerické výpočty. V těchto grafech, se nacházejí uzly (node) reprezentující funkci, nebo výpočet, který má být proveden a hrany grafu spojující uzly představují data, která mezi nimi procházejí. V TensorFlow jsou data v podobě vícerozměrných polí označované jako tenzory. [2]

V této diplomové práci je využitá knihovna TensoFlow ve verzi 2.4. Práce v knihovně TensoFlow před verzí 2.0 představovala vytvoření tzv. statický výpočetních grafů. To znamená, nejdříve definovat výpočetní graf a následně vyhodnotit pro dostání výsledku. TensoFlow od verze 2.0 má přidanou podporu tzv. eager execution. Tento nový přístup umožňuje vytváření dynamických grafů. Vytváří se imperativní programovací prostředí, které vyhodnotí operace okamžitě. Operace mají konkrétní návratovou hodnotu namísto konstruování statického grafu. Eager execution výrazně zjednodušuje práci a ladění grafů. Eager execution přináší i nevýhody pomalejších výpočtů a celkové nižší efektivnosti. V této práci, ale byl téměř výhradně využit právě eager execution pro intuitivnost a možnost ladění při vytváření dynamických grafů. Jedinou výjimkou je využití statických grafů u generátoru dat tf.data.Dataset (viz 3.2.1), který nemá podporu eager execution. [2][9]

TensorFlow podporuje oficiálně operační systémy Ubuntu a Windows. TensorFlow je oficiálně poskytovaný v Python a C++ APIs. V této práci je využitý pouze operační systém Windows, kde je v případě TensorFlow podpora pouze Python 3. V případě Ubuntu je podpora i pro Python 2. TensorFlow umožňuje běh na procesoru, tak i na dedikovaných grafických kartách od společnosti Nvidia. [2]

3.2.1 Generátor dat

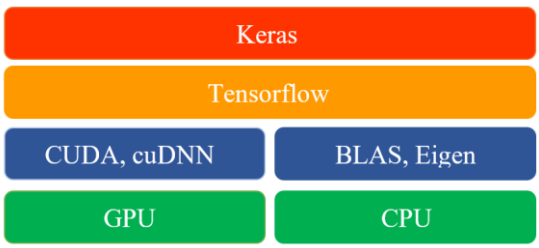
V rámci trénování neuronových sítí je nutné dodávat v pravidelných intervalech data. Skupině funkcí, které se starají o načítaní dat z disku, jejich předzpracování a následné dodání dat trénovací sekvenci, se označují jako generátor dat. Generátor dat bývá často úzkým hrdlem, který zpomaluje celý proces trénování z důvodu práce v jednom vlákně a výpočetně náročného předzpracování. Proto v této diplomové práci byl využit soubor funkcí tf.data.Dataset z knihovny TensorFlow, tyto funkce jsou vysoce optimalizované pro velký objem dat a náročné předzpracování dat.

Soubor funkcí tf.data.Dataset nedefinuje konkrétní operace pro načtení a ani předzpracování dat. Funkce slouží pouze pro vytvoření sekvence operací, ale jednotlivé operace jako načtení dat z disku, škálování obrazových dat, augmentace apod., je nutné doimplementovat. Funkce tf.data.Dataset například zajišťují rozložení jednotlivých operací do vláken, před načítání dat do bufferu, náhodné zamíchání dat, seskupení dat (batch) a další. [2]

Jistou nevýhodou při vytváření sekvence pomocí tf.data.Dataset je, že každá funkce a operace musí fungovat jako statický graf, a to způsobuje pomalejší vývoj jednotlivých funkcí kvůli nemožnosti klasického ladění. [2]

3.3 Keras

Jedná se o vysoko úrovňový API, které je součástí TensorFlow od verze 2.0. Výhodou Keras API je, že umožňuje jednodušší a rychlejší vývoj oproti nízko úrovňovému TensorFlow. Přestože Keras zjednodušuje práci s nízko úrovňovým frameworkem, tak si stále zachovává flexibilitu, modularitu, snadnou rozšiřitelnost. Na Obr. 3.8 je ukázka hierarchie Keras a TensorFlow, ze které je zřejmé, že pro využití grafické karty je nutná dostupnost knihoven Nvidia CUDA a cuDNN. Knihovny se používají pro tenzorové operace. [2]



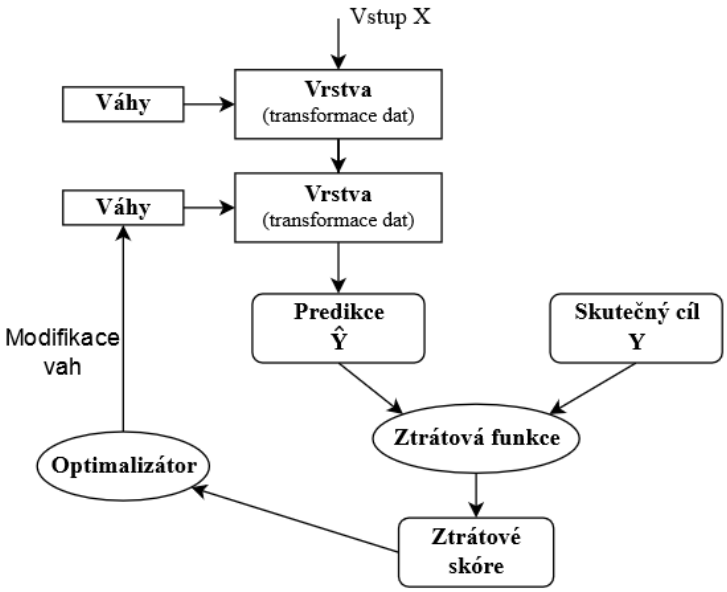
Obr. 3.8 *Struktura softwaru a hardwaru pro Keras* [4]

3.3.1 Základní moduly v Keras

Neuronové sítě se skládají z několika základních bloků zobrazených v diagramu (Obr. 3.9). Jsou to vrstvy, ztrátová funkce a optimalizátor.

Základním stavebním blokem neuronových sítí je vrstva, která představuje modul pro zpracování dat. Vrstvu je možné chápat jako datový filtr, to znamená, že extrahují reprezentace ze vstupních dat. Na vstupu může přijímat jeden a více tenzorů a na výstupu má obdobně jeden až více tenzorů. Vrstvy uchovávají svoje naučené znalosti ve formě vah, což představuje jeden, nebo více tenzorů se stochastickým sestupem gradientu. Existují i vrstvy bez vah, které obsahují pouze matematickou operaci bez možnosti učení, například převzorkování tenzoru. [4]

Vzájemným propojením jednoduchých vrstev vytvářejí hluboké sítě, nebo také model hlubokého učení, které následně realizují tzv. destilaci dat. Jednotlivé vrstvy můžou být například 2D konvoluční vrstvy, plně propojené vrstvy (Dense layer) apod. Důležité je vhodně kombinovat vrstvy, tak aby byly kompatibilní. Každá vrstva přijímá data určitého tvaru a na výstupu často vrací tenzor jiného tvaru. [4]



Obr. 3.9 *Struktura a vztahy mezi základní stavebními bloky NN [4]*

Na Obr. 3.10 je ukázka jednoduchého modelu se třemi vrstvami. První vrstva definuje velikost vstupního tensoru. Druhá vrstva je plně propojená vrstva s váhami. A poslední vrstva je výstupní, definující tenzor o velikosti deseti hodnot (například 10 tříd).

```
1.  from Keras import models
2.  from Keras import layers
3.
4.  input_tensor = layers.Input(shape=(784,))
5.  x = layers.Dense(32, activation='relu') (input_tensor)
6.  output_tensor = layers.Dense(10, activation='softmax')(x)
7.
8.  model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Obr. 3.10 *Ukázka vytvoření jednoduchého modelu*

Model hlubokého učení (dále zkráceně model) může být v několika topologiích. Nejjednodušší je lineární zásobník vrstev, to znamená jeden vstup a jeden výstup. Další může být síť se dvěma větvemi, nebo i více větvené sítě. Zvolená topologie definuje tzv. prostor hypotéz. To znamená, že volbou topologie se dojde k omezení prostoru možností (hypotéz). Ve skutečnosti vznikne série tenzorových operací mapující vstupní data na výstupní data. Trénováním potom dochází k hledání vhodných tenzorových vah, které transformují vstup na požadovaný výstup. [4]

Po definování vhodné architektury je potřeba vybrat ztrátovou funkci, kde cílem je její hodnotu minimalizovat během trénovaní. Model s více výstupy může mít pro každý výstup vlastní ztrátovou

funkci. Proces gradientního sestupu musí být založen pouze na jediné skalární hodnotě, a proto je nutné zprůměrovat všechny hodnoty ze ztrátových funkcí do jediné skalární hodnoty. [4]

Poslední částí je optimalizátor, který určuje, jak bude síť aktualizovaná na základě ztrátové funkce. Implementuje variantu stochastického gradientního sestupu. V Keras API se před trénováním modelu určí ztrátová funkce a optimalizátor a provede se kompilace modelu (Obr. 3.11) [4]

```
1. from Keras import optimizers
2. model.compile(optimizer=optimizers.RMSprop(lr-0.001), loss='mse', metrics=['accuracy'])
```

Obr. 3.11 *Kompilace modelu*

Poslední krok je spuštění samotného trénování modelu s definováním trénovacích a cílových dat (Obr. 3.12). Dále je potřeba zvolit velikost dávky dat (batch size), který určuje počet dat, které se nahrají do paměti grafické karty a zpracují se najednou. Velikost dávky je omezena velikostí operační paměti grafické karty. Poslední parametr je počet epoch, který definuje kolikrát má trénovací algoritmus projít celý trénovací dataset.

```
1.    model.fit(input_tensor,target_tensor,batch_size=128, epochs=10)
```

Obr. 3.12 *Trénování modelu*

3.4 Albumentations

V rámci trénování neuronových sítí je nutné zajištění větší robustnosti modelu a zabránění přeučení (neboli overfitting). Toho je možné dosáhnout dvěma způsoby, první je trénovat neuronovou síť na velkém objemu dat. Tento přístup, ale přináší problémy jako nedostupnost většího množství dat, časově náročné anotovat další vzorky trénovacích dat atd. Druhým přístupem je tzv. augmentace dat, což představuje rozšíření trénovací sady pomocí řady matematických operací na trénovacích datech.

Pro augmentaci dat je možné zvolit několik přístupů, první je naprogramování vlastních funkcí, ale to vyžaduje dlouhý vývoj a optimalizace, a proto nebyl využit tento přístup. Druhou možností je zvolit existující knihovny. Samotné TensorFlow má funkce pro augmentaci obrazu, ale výběr funkcí je značně omezený. Nakonec byla zvolena Python knihovna Albumentations. Její výhoda je široká škála transformací obrazu, rychlost v porovnaní s ostatními knihovnami a také kompatibilita s funkcemi z TensorFlow Dataset využité pro trénovací sekvenci. [1]

```
1. transforms = A.Compose([
2.     A.VerticalFlip(p=0.4),
3.     A.HorizontalFlip(p=0.4),
4. ])
```

Obr. 3.13 *Ukázka augmentační sekvence*

V knihovně se nepřistupuje k jednotlivým transformacím, ale na začátku trénování se vytvoří tzv. augmentačním sekvence. Na Obr. 3.13 je ukázka jednoduché augmentační sekvence. Obsahuje transformaci horizontálního a vertikálního překlopení. Každá z transformací má možnost nastavení

pravděpodobnosti od 0 do 1. Samozřejmě je možné vytvářet mnohem komplexnější sekvence, kde se například seskupí několik transformací a při provedení augmentace se zvolí ze skupiny pouze jedna. Dále je v knihovně dostupné další transformace jako je rotace, zvýšení ostrosti, šum, úprava jasu, kontrastu a další. [1]

```
1.  def aug_fn(self, image, mask):
2.      data = {"image": image, "mask": mask}
3.      aug_data = self.transforms(**data)#provedeni augmenatcni sekvence
4.      aug_img = aug_data["image"]
5.      aug_mask = aug_data["mask"]
6.      aug_img = tf.cast(aug_img, tf.float32)
7.      aug_mask = tf.cast(aug_mask, tf.float32)
8.
9.      return aug_img, aug_mask
```

Obr. 3.14 *Provedení augmentace na obrazu a masky zároveň*

Vytvořená augmentační sekvence pomocí knihovna Albumentations, je využitá v generátoru dat, který je popsaný v podkapitole 3.2.1. V podkapitole 3.2.1 je zmíněné, že funkce musí podporovat zpracovaní ve statické výpočetním grafu, ale knihovna Albumentations nemá podporu pro statické grafy. Avšak knihovna tf.data.Dataset má speciální funkci pro zapouzdření funkcí nefungující jako statický graf. Na Obr. 3.15 je ukázka této funkce tf.numpy_function, u které je nutné definovat vstupní parametry a datový typ výstupních parametrů. Nevýhoda využití této funkce je snížení efektivnosti při zpracovaní. [1] [2]

```
1.  image, mask = tf.numpy_function(func=self.aug_fn, inp=[image,mask], Tout=[tf.float32,tf.float32])
```

Obr. 3.15 *Zapouzdření augmentační sekvence v tf.data.Dataset*

3.5 LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) je programovací a vývojové prostředí, které je vyvinuté společností National Instrument (dále NI). Překlad akronymu LabVIEW je "laboratorní pracoviště virtuálních přístrojů". Tato platforma využívá tzv. G-jazyk (grafický jazyk), který je využíván pro vytváření programů měření a analýzu signálů, řízení a vizualizaci. Virtuální instrumentace umožňuje rychlý návrh nových aplikací a provádění změn v konfiguraci aplikace. [1]

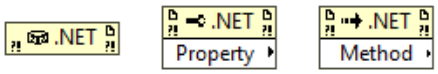
Společnost NI nabízí kromě vývojového prostředí se základními knihovnami i celou řadu modulů umožňující rozšířit funkce daného programovacího prostředí. V této práci je využitý Vision Development Module (dále VDM), který nabízí široku podporu funkcí pro zpracování obrazu.

Součástí platformy LabVIEW je celá řada knihoven. Budou zde uvedeny tři zásadní knihovny využité v této práci.

3.5.1 Knihovna .NET

LabVIEW má knihovnu funkcí pro práci s .NET. Na Obr. 3.16 jsou ukázané tři základní funkce. První je funkce Constructor Node, která umožňuje otevření reference na třídu s příslušnými metodami. Třída musí být zkompilovaná ve formátu DLL (Dynamic-link library). Po otevření reference na třídu je možné přistoupit k jednotlivým metodám pomocí funkce Invoke Node. Výstupem z metody je často

struktura, ta je v případě Ivoke Node v podobě reference na objekt. Pro přístup k jednotlivým prvkům struktury je potřeba využít funkce Property Node. Po dokončení práce s otevřenou třídou je nutné uzavřít referenci. [17]

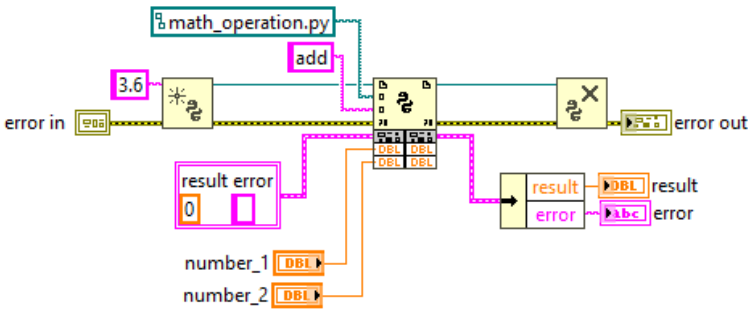


Obr. 3.16 *Constructor , Property a Invoke Node*

3.5.2 Knihovna Python

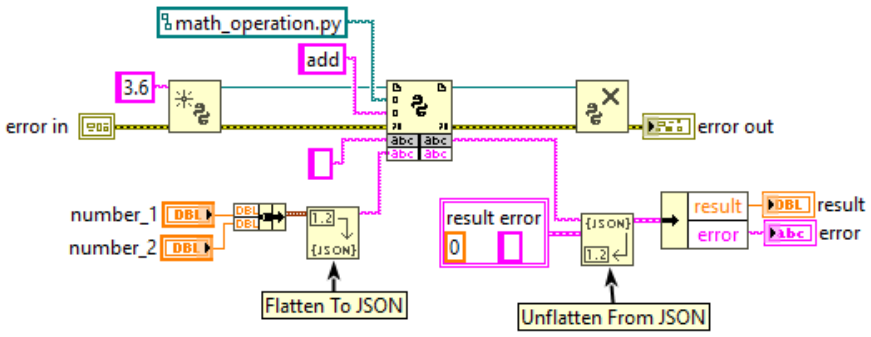
Druhá důležitá knihovna je Python. Tato knihovna je učená pro spouštění metod vytvořených v jazyce Python ve verzi 2.7, nebo 3.6. Nutností je před využitím této knihovny naistalovat do operačního systému Python v jedné už zmíněných verzí. [17]

Na Obr. 3.17 je ukázka jednoduchého příkladu funkce pro sčítání dvou čísel pomocí Python funkce. První blok je funkce Open Python Session, která vytvoří referenci na relaci zvolené verze Pythonu, ve které se budou spouštět metody. Druhý blok je funkce je pro samotné spuštění metody. Parametry funkce je cesta k souboru s příponou PY, ve kterém jsou definovány metody. Druhý parametr je název metody, kterou chci spustit, v tomto případě *add.* Třetí parametr je konstanta návratového typu, v tomto případě cluster. Následující vstupy přestavují vstupní parametry funkce. Návratová hodnota je ve formě clusteru, ekvivalentem na straně Pythonu je datový typ *tuple*, který musí být složený ze stejných datových typů a pořadí jako je definovaný v LabVIEW. [17]



Obr. 3.17 *Příklad na sečtení dvou čísel pomocí Python metody*

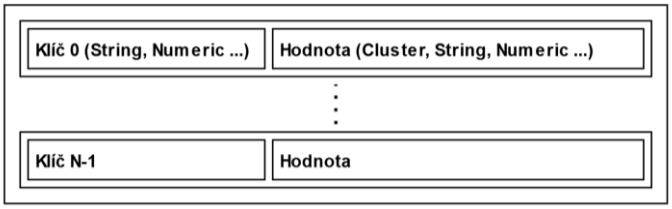
U popsaného příkladu na Obr. 3.17 je nutné definovat každý vstupní parametr metody, a to se stejným datovým typem. Tento přístup není příliš vhodný v případě velkého počtu vstupních parametrů a také v případě vývoje, kdy se můžou postupně přidávat další parametry. Z těchto důvodů je zvolený odlišný přístup, který je ukázaný na Obr. 3.18. Na straně LabVIEW jsou veškeré vstupní parametry sdruženy do jediného clusteru a následně převedeny na JSON string pomocí funkce Flatten to JSON, výstupem je string. JSON je textový formát pro reprezentaci strukturovaných dat. Tento string je využitý jako vstupní parametr metody, a kde na straně Pythonu se převede tento JSON string na datovou strukturu Dictionary (slovník).



Obr. 3.18 *Příklad na sečtení dvou čísel pomocí Python metody – druhý přístup*

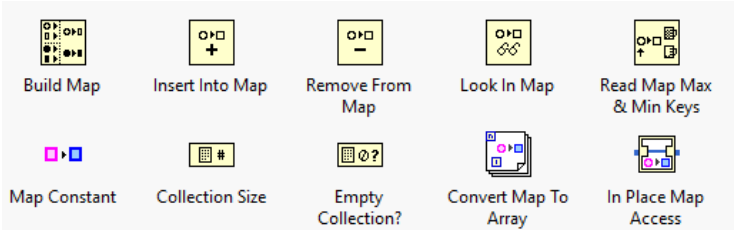
3.5.3 Knihovna Collection

Knihovna Collection se skládá ze dvou částí: Maps a Sets. První část Maps je možné přirovnat k datové struktuře Dictonary, která je dostupná např. v jazyce C#. To znamená, že je to struktura (Obr. 3.19), která obsahuje libovolný počet hodnot s libovolným datovým typem podobně jako je pole, ale každá z těchto hodnot je mapovaná na zvoleném klíči. Klíč může být také libovolného datového typu, ale po zvolení je nutné pro vytvořenou mapu využívat pouze zvolený datový typ. Stejným omezením je pro zvolený datový typ hodnoty. [17]



Obr. 3.19 *Struktura mapy*

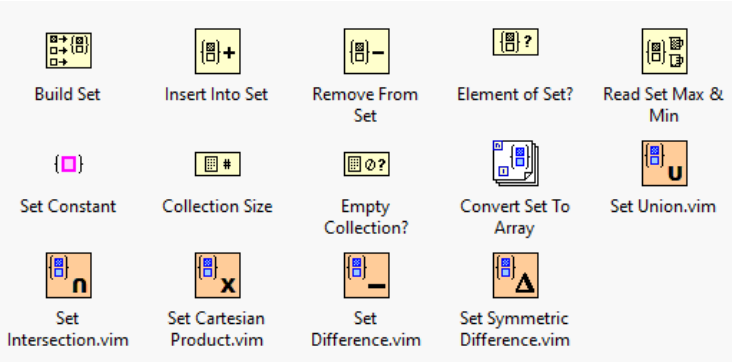
Na Obr. 3.20 je ukázka všech dostupných funkcí pro práci s mapami. Funkce umožňují vkládání nových hodnot se zvoleným klíčem, smazání hodnoty podle klíče, hledání hodnoty podle klíče. Pokud je nutné převést mapu na pole, tak je to možné pomocí smyčky FOR, kde pomocí auto-indexování se provede převod mapy na pole s hodnotami. [17]



Obr. 3.20 *Funkce pro práci s mapami* [17]

Druhou částí jsou tzv. Sets. Jedná se o podobnou strukturu jako je pole. To znamená, že umožňuje definovat set pro libovolný datový typ. Rozdíl oproti poli, je že každá hodnota je unikátní. Například pokud se vytvoří set s datovým typem Numeric a provede se přidání hodnoty 7, tak při opětovné snaze o vložení hodnoty 7 se neprovede vložení, protože hodnota už je obsažena v daném setu. Další rozdíl oproti poli je, že set udržuje posloupnost vložených hodnot. [17]

Výhodou využití setů oproti polím je, že umožňují celou řadu operací (Obr. 3.21). Dostupné je např. sjednocení dvou setů, přičemž se ve sjednoceném setu zachovává unikátnost každé hodnoty a nevznikají tak duplikátní hodnoty. Dále je podpora rozdílu, průniku dvou setů a další. [17]



Obr. 3.21 *Funkce pro práci se sety* [17]

3.6 Ostatní nástroje a hardware

Kromě už popsaných nástrojů a knihoven bylo pro vývoj využity další softwarové nástroje. Pro vývoj aplikací .NET v jazyce C# bylo využité vývojové prostředí Visual Studio 2019. Pro vývoj aplikací v jazyce Python byl využitý Pycharm 2020 Community edition.

Veškerý práce v Cognex DLS a vývoj v knihovnách TensorFlow a Keras, byl vyvíjená na počítačové sestavě s dedikovanou grafickou kartou Nvidia GeForce RTX2080 Ti 11 GB, dále sestava má celkem 16 GB operační paměti a procesor Intel Core i7-8700 3,2GHz.

4 Praktická realizace

V následující kapitole je popsaný postup praktické tvorby diplomové práce. Na začátku byl zvolený firemní dataset SMD součástek (4.1). V úvodu kapitoly 3 bylo popsané rozhodnutí srovnat dva přístupy. První je využití komerčního nástroje Cognex DLS a druhý přístup je využít open source knihovnu TensorFlow. Cognex DLS byl zároveň využitý pro anotaci datasetu. Na anotovaném dataset byl následně natrénovaný model v prostředí Cognex DLS (4.3.1). Pro natrénovaní vlastního řešení v TensorFlow bylo potřebné získat anotace z programu Cognex DLS. Ten, avšak neumožňuje export anotací, ale umožňuje doprogramovat vlastní plugin v .NET (4.3.2). Po vytvoření pluginu a exportu anotací bylo možné vytvořit trénovací aplikaci v jazyce Python s knihovnou TensorFlow a Keras (4.5). Posledním krokem bylo vytvoření testovací aplikace pro evaluaci modelů, která se zároveň stará o výpočet evaluačních metrik (4.6). Prostředí Cognex DLS poskytuje také evaluační metriky na základě, kterých je možné zhodnotit natrénovaný model. Vzhledem k tomu, že není možné zajistit shodnost výpočtu metrik v Cognex DLS a výpočtu metrik ve vlastní testovací aplikaci, tak bylo nutné vyhodnotit Cognex model (4.3.3) a výsledky z predikce importovat do vlastní testovací aplikace, kde se následně vypočítaly metriky pro Cognex model. Testovací aplikace byla vytvořena kombinací kódu v Pythonu a hlavní část ve vývojovém prostředí LabVIEW.

Před samotným popisem praktické realizace je nutné specifikovat směr, kterým se tato diplomová práce ubírá. Oblast strojového učení (v kontextu zpracování obrazových dat) se dělí na tři základní úlohy. Úlohami jsou lokalizace, klasifikace a segmentace. V této práci je hlavní zaměření na segmentaci, jejímž cílem je ohraničení hledaného objektu v dodaném obrazu, a to na pixelové přesnosti. To znamená, model s architekturou pro segmentaci zjišťuje u každého pixelu dodané obrazu, k jaké třídě náleží. Segmentační úlohu lze dále dělit na instanční a sémantickou. Instanční segmentace rozlišuje jednotlivé instance objektů v obraze. Sémantická segmentace bere všechny hledané objekty jako jednu instanci. Tato práce je zaměřená právě na sémantickou segmentaci a částečně i na klasifikační úlohu.

4.1 Dataset SMD součástek

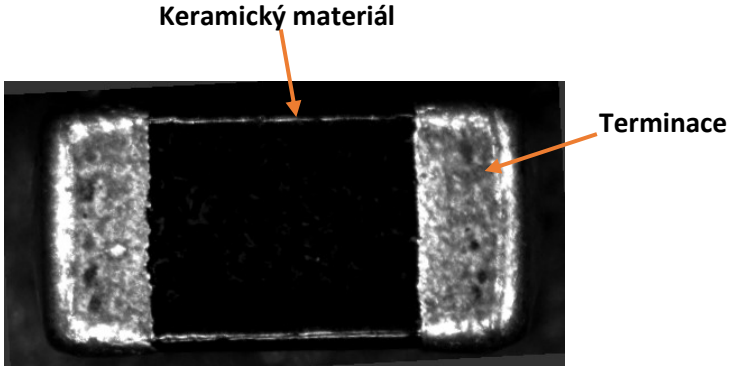
V rámci zadání diplomové práce je ověření zvolených technik strojového učení. Pro ověření je nutné otestovat tyto techniky na datasetu snímku, který se bude týkat vyhodnocení defektní kusů. Jelikož se jedná o diplomovou práci zadanou firmou, tak bylo umožněno využít firemní dataset SMD součástek. V Tab. 4.1 je ukázané, jak byl rozdělený dataset, který obsahuje celkem 2000 snímků. Trénovací sada slouží k natrénovaní modelu. Testovací dataset je potom určeno pro ověření, že natrénovaný model je dostatečně robustní, a hlavně schopný predikce na doposud neviděných datech.

Tab. 4.1 *Dataset SMD součástek*

Dataset	OK	NOK	Celkem
trénovací	397	403	800
testovací	656	544	1200

Na Obr. 4.1 je ukázaný jeden snímek z použitého datasetu. Snímky byly nasnímány na různých kamerách, a potom byly oříznuté. Vzhledem k tomu, že byly snímky nasnímané na různých kamerách,

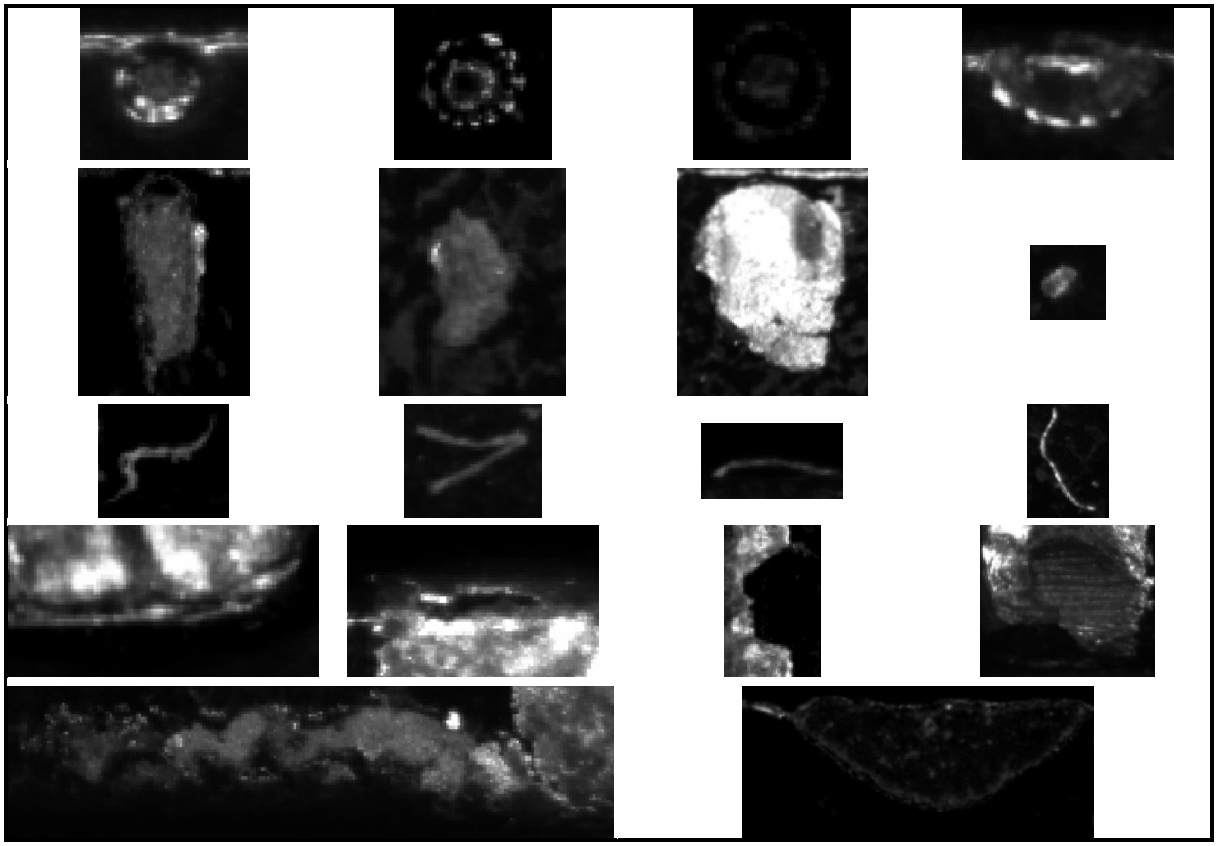
tak výsledně oříznuté snímky nejsou jednotně velké. Výška se pohybuje mezi 250 až 262 pixely a šířka snímku je v rozmezí 495 až 510 pixelů.



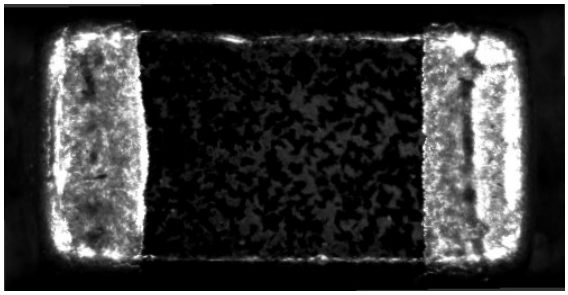
Obr. 4.1 *Snímek SMD součástky*

V Tab. 4.2 je ukázka častých defektů na SMD součástkách. Většina defektů se nachází na keramickém materiálu dielektrika. První řádek přestavuje díry v keramickém materiálu, až na poslední snímek, které přestavuje úlomek keramiky na hraně SMD součástky. Druhá řádek jsou nečistoty na keramické materiálu, nejčastěji se jedná o cín. Na třetím řádku jsou ukázané vlákna na keramice. Čtvrtý řádek jsou praskliny, nebo chybějící části v terminaci. Poslední řádek jsou defekty na keramickém materiálu, které jsou způsobené narušením vrchní keramiky. V datasetu jsou i atypické defekty, které můžou být reprezentovány i pouze jedním snímkem.

Tab. 4.2 *Ukázka častých defektů*



V rámci datasetu SMD součástek existují různé odchylky, které nejsou považovány za defekty. Jednou z nejčastějších odchylek je tzv. mramorování, které se projevuje na keramické materiálu dielektrika. Ukázka této odchylky je na Obr. 4.2.



Obr. 4.2 *SMD součástka s mramorováním*

V této práci se často operuje s pojmem anotace. Synonymem pro slovo anotace je slovo označení. V oblasti strojového učení se, ale častěji používá právě slovo anotace. Anotací může být například ohraničující rámeček (bouding box), třída anebo segmentační mapa. Anotace v případě třídy je myšlená klasifikace snímků podle definovaných tříd (např. třídy OK a NOK). V případě že se využívají pouze dvě třídy, tak se tento typ úlohy označuje jako binární klasifikace. Segmentační mapy jsou zpravidla obrázky, které jsou ve velikosti anotovaných snímků. Segmentační mapy jsou určeny pro označení objektů ve snímcích na pixelové úrovni. To znamená, že každému pixelu ve snímku je přiřazena třída. U segmentace je vždy přítomna třída pozadí (backgrounds). V případě více jak jedné třídy se využívají barevné segmentační mapy, kde jednotlivé barvy určují konkrétní třídu. Když se operuje pouze s jednou třídou a plus samozřejmě pozadím, tak se tato varianta označuje jako binární segmentace. Segmentační mapy pro použití binární segmentace jsou ve formátu obrázku ve stupních šedi, respektive třída pozadí nabývá hodnoty 0 a uživatelem zvolená třída nabývá hodnoty 255. V této diplomové práci je využita binární segmentace a zvolená třída je označená jako defekt, dále se pracuje i s binární klasifikací, kde třídy jsou OK a NOK. Segmentační mapy byly získané exportem z Cognex DLS (4.3.2), exportovány byly pouze pro NOK snímky, protože u OK snímku jsou pixely segmentační mapy pouze hodnota 0.

4.2 Evaluační metriky

Evaluační metrika je obecný pojem využívaný ve strojovém učení, který může zahrnovat mnoho metrik (parametrů) na základě kterých se hodnotí kvalita predikce modelu. Metriky se zpravidla počítají jako vztah mezi skutečným cílem a predikcí modelu. Výběr metrik se odvíjí od typu úlohy jako klasifikace, detekce objektu anebo segmentace. V této diplomové práci je hlavní zaměření na binární klasifikaci a segmentaci, proto v následujících podkapitolách budou vysvětleny metriky týkající se této oblasti.

Konktrétně jsou vysvětlený metriky pro binární klasifikace jako je matice záměn (confusion matrix), accuracy a AUC-ROC. A dále metriky pro segmentaci jsou IoU (Intersectio over Union), F1, Recall a Precision.

Pro pochopení použitých metrik je potřeba vysvětlit pojmy True Positive (dále jen TP), False Positive (dále jen FP) a nakonec False Negative (dále jen FN). Všechny čtyři pojmy se částečně liší v jejich interpretaci vzhledem k typu úlohy jako je v tomto případě binární klasifikace a segmentace.

Proto budou nejdříve vysvětleny tyto pojmy vzhledem k binární klasifikaci a následně ve vztahu k segmentaci.

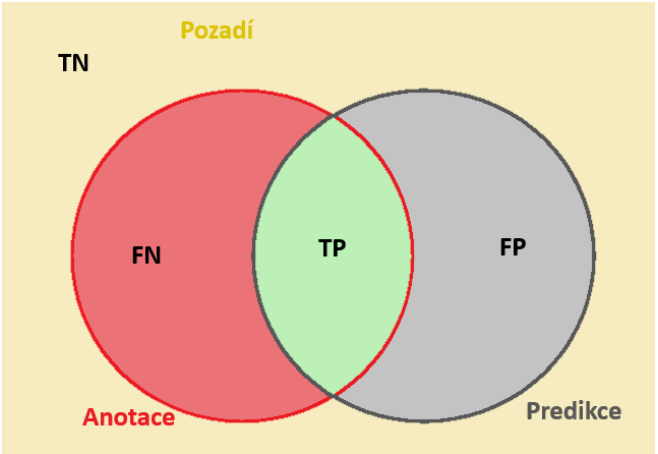
Binární klasifikace je typ úlohy, kde se klasifikuje snímek do dvou kategorií, v této práci to jsou třídy OK a NOK. Model na svém výstupu často poskytuje pravděpodobnost v podobě skalární hodnotu od 0 do 1. Pokud například namapujeme třídu OK jako 0 a třídu NOK jako 1, tak potom je nutné definovat práh T (threshold). Na základě prahu se provede rozhodnutí, jestli výstupní hodnota z modelu predikuje první třídu OK, nebo druhou NOK třídu. Potom pojmy TP, TN, FP a FN jsou definovány v Tab. 4.3. Pojem TP a TN je pro případ, kdy se predikovaná třída rovná skutečné cílové. Na druhou stranu pojmy FP a FN jsou pro případ, kdy se nerovnají třídy.

$$predikovaná\ třída = \begin{cases} OK & klasifikační\ skóre > T \\ NOK & klasifikační\ skóre \leq T \end{cases} \tag{4.1}$$

Tab. 4.3 *Roztřídění výsledků do TP, TN, FP a FN*

$třída_{skutečná}$	$třída_{predikovaná}$	Kategorie
OK	OK	TP
OK	NOK	FN
NOK	NOK	TN
NOK	OK	FP

Další typ úlohy je binární segmentace, která je obdobou binární klasifikace popsané v předešlém odstavci. Rozdíl je, že na výstup z modelu je predikční mapa v podobě 2D pole o velikosti vstupního snímku. Každá hodnota v predikční mapě je skalární hodnota od 0 do 1, které udává pravděpodobnost výskytu definované třídy. V případě této diplomové práce je mapovaná hodnota 0 jako pozadí a hodnota 1 jako defekt. Na Obr. 4.3 je vysvětlen význam TP, TN, FP a FN pro segmentaci. Jedná se o překryv dvou obrázků. První obrázek je zastoupený kruhem anotace, který představuje označený defekt a pozadím. Neoznačená oblast je označovaná jako pozadí (backgrounds). Druhý obrázek kruh predikce představuje výstupní predikci z modelu spolu s pozadím. Překryvem těchto dvou obrázků se získají oblasti TP, TN, FP a FN.



Obr. 4.3 *Ukázka TP, TN, FP a FN pro segmentaci*

4.2.1 **Matice záměn a Accuracy**

Matice záměn (Confusion Matrix) je matice, která zobrazuje kvalitu predikcí modelu. Je to metrika, která se velice často používá u hodnocení kvality klasifikátoru. Matice (Tab. 4.4) je rozdělena na dva směry. První jsou řádky (Anotované), které představují pravé cílové hodnoty. A druhý směr sloupce (Predikované) jsou naopak modelem predikované hodnoty. V případě ideálního modelu bychom dostali hodnoty pouze na diagonále TP a TN. Když ale nastane situace, kdy se neshoduje třída cílové hodnoty s predikovanou, tak se zapíše do buněk označených FP anebo do FN (Tab. 4.4).

Tab. 4.4 *Obecná matice záměn pro binární klasifikátor*

		Predikované	
	Třídy	**OK**	**NOK**
Anotované	**OK**	TP	FN
	NOK	FP	TN

Druhá metrika, která souvisí s maticí záměn je Accuracy. Jedná se o parametr, který se počítá z matice záměn. Výpočet je ukázaný ve vzorci (4.2) a definuje přesnost predikce klasifikátoru. Tato metrika může být za určitých okolností zavádějící, tento fenomén se označuje jako Accuracy Paradox. Paradox může nastat například pokud jsou nevyvážené třídy, tak může dojít k tomu, že může být vysoká Accuracy, ale i přesto se může jednat o špatný klasifikátor.

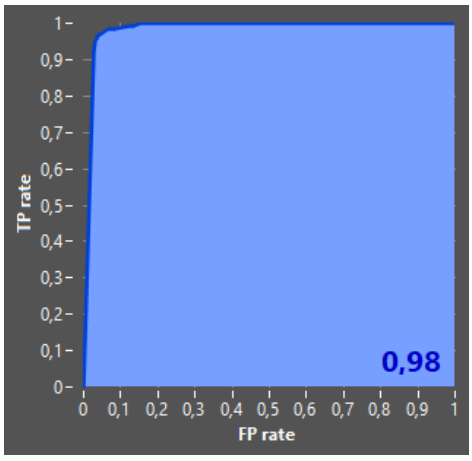
$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{4.2}$$

4.2.2 **AUC-ROC**

Metrika, která hodnotí kvalitu binárního klasifikátoru. Název této metriky se skládá ze dvou částí AUC (Area Under Curve – Plocha pod křivkou) a ROC (Receiver Operator Characteristic – Operační charakteristika přijímače). Druhá část ROC představuje křivku, ze které se počítá plocha pod křivkou pro získání jediné skalární hodnoty od 0 do 1, kde hodnota 1 je pro ideální klasifikátor.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN} \tag{4.3}$$

Na Obr. 4.4 je ukázka ROC křivky a zároveň v rohu obrázku je hodnota $AUC = 0{,}98$. Jak už bylo zmíněné v úvodu podkapitoly 4.2, tak klasifikátor poskytuje na výstupu klasifikační skóre od 0 do 1 a u kterého je nutné zvolit prahovou hodnotu T pro rozhodnutí kterou třídu klasifikátor predikuje. Při výpočtu ROC křivky se pracuje právě s tímto prahem, který se nastavuje 0 do 1 s definovaným krokem a při nastavení daného kroku se vypočítají TPR (TP rate) a FPR (FP rate) podle rovnice (4.3). Jednotlivé hodnoty TPR a FPR při určitém prahu představují body v ROC křivce. Hodnota AUC se potom spočítá integrování vypočítané plochy ROC křivky.



Obr. 4.4 *Ukázka ROC křivky*

4.2.3 IoU, F1 , Recall a Precision

Metriky popsané v této podkapitole hodnotí kvalitu segmentace, všechny se počítají z 2D polí představující anotaci a predikci, ale výsledné metriky jsou skalární hodnoty. Ve vzorcích (4.3 až (4.7 je použité označení A pro anotaci a P pro predikci. První metrikou je IoU (Intersection over Union), která představuje výpočet překryvu anotace s predikcí. Další metrika F1 (také označovaná jako Dice coefficient) je obdobou IoU, to znamená, že se počítá překryv anotace s predikcí, ale rozdíl je, že u F1 je kladený větší důraz na překryv, než je tak u IoU.

$$IoU = \frac{|A \cap P|}{|A \cup P|} = \frac{TP}{TP + FN + FP} \tag{4.4}$$

$$F1 = \frac{2 \cdot |A \cap P|}{|A| + |P|} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP} \tag{4.5}$$

Metrika Recall dává důraz na oblast FN. To je oblast, která byla anotovaná, je nebyla model predikovaná. Naproti tomu metrika Precision dává důraz na oblast FP. FP je oblast, která predikovaná model, ale není anotovaná. Z definice metrik Recall a Precision vyplývá, že metrika Recall je velice důležitá při detekci defektů. Je mnohem důležitější zachytit všechny defekty (vyšší Recall) na úkor nižší přesnosti (nižší Precision).

$$Recall = \frac{|A \cap P|}{|A \cap P| + |A - P|} = \frac{TP}{TP + FN} \tag{4.6}$$

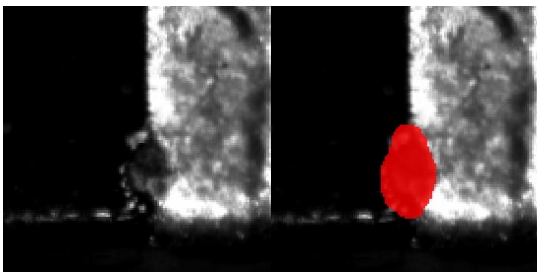
$$Precision = \frac{|A \cap P|}{|A \cap P| + |P - A|} = \frac{TP}{TP + FP} \tag{4.7}$$

4.3 Cognex DLS

4.3.1 Trénování modelu

Na začátku práce s Cognex DLS bylo nutné vytvořit projekt, do kterého se naimportovaly snímky. Druhým krokem bylo zvolení nástroje ze čtyř dostupných: Green Classify, Blue Locate, Blue Read a Red Analyze (bližší popis nástrojů 3.1). Zvolený byl nástroj Red Analyze, který umožňuje

anotovat konkrétní defekty, a tudíž nejvhodnější pro tento typ úlohy. Po zvolení nástroje se provedla anotace, a to na dvou úrovních. První bylo rozdělení snímků na OK a NOK kusy. A druhá úroveň představovala anotaci konkrétních defektů na NOK snímcích (Obr. 4.5).



Obr. 4.5 *Ukázka defektu bez a s anotací v Cognex DLS*

Po dokončení bylo provedeno nastavení trénování a modelu. Tento proces byl iterativní, to znamená, že se postupně upravovali parametry pro dosažení optimálních výsledků (v Tab. 4.5 a Tab. 4.6). Bližší popis jednotlivých parametrů je v podkapitole 3.1.1. Zvolený přístup bylo využití režimu Supervised (s učitelem), druhý režim Unsupervised je vhodnější pro úlohy, kde se nachází souvislé struktury např. dřevěné desky, tkanina apod. Lepší výsledky byly dosaženy zvolením velkého modelu a zvolení Feature size 30 pixelů. Pro natrénování modelu bylo nastaveno celkem 100 epoch.

Tab. 4.5 *Nastavení trénování v Cognex DLS*

Mode	Supervised
Network model	Large
Feature size (pixel)	30
Sampling density	6
Threshold	0,52

V Tab. 4.6 je popis nastavení augmentační sekvence. Z nastavení je zřejmé, že se jednalo pouze o lehkou augmentaci. Při nastavení silnější augmentace (např. větší úhel rotace, škálování apod.) se zhoršovaly výsledky. Vzhledem k tomu, že snímky jsou poměrně unifikované (např. se příliš nemění jas, rotace apod.), tak není nutné volit silnější augmentaci.

Tab. 4.6 *Nastavení augmentace snímků*

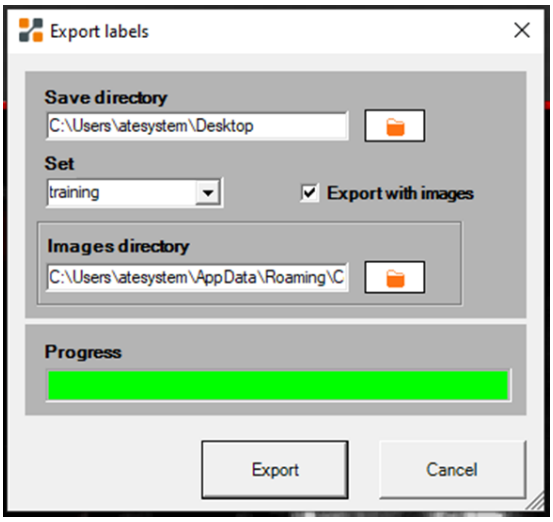
Augmentace
Rotace [-1°; +1°]
Vertikální překlopení
Horizontální překlopení
Jasnost 1 %
Kontrast 1 %
Rotace [-1°; +1°]

Vzhledem k tomu, že Cognex DLS je uzavřený software, tak zde není možné dále popsat využité architektury a bližší přístup k samotnému trénování.

4.3.2 Export anotací

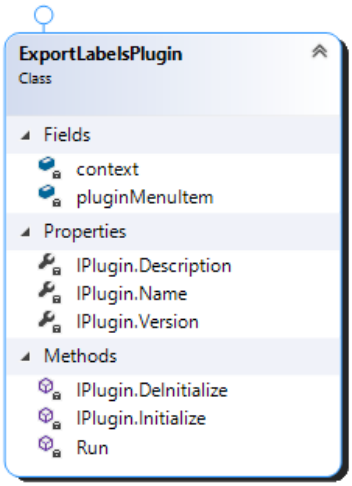
Pro natrénování vlastních neuronových sítí bylo nutné získat anotace z Cognex DLS. Tento software nemá dostupnou funkci pro export anotací, ale umožňuje doprogramovat si vlastní plugin. Vytvořený plugin má možnost rozšířit funkce softwaru. Jednou z podporovaných možností je import a export anotací z vytvořeného projektu. Pro naprogramování plugin poskytuje Cognex .NET knihovnu. Vytvořený kód je nutné zkompilovat do DLL knihovny, kterou je potom možné přidat do Cognex DLS.

Na Obr. 4.6 je ukázka grafického rozhraní vytvořeného pluginu. Na začátku je potřeba zvolit složku ve které se automaticky vytvoří hlavní adresář s názvem projektu. Dalším parametrem je Set, který umožní zvolit konkrétní snímky, které se mají vyexportovat. *Set* představuje v terminologii Cognex DLS formu seskupení snímků pod uživatelem zvoleným názvem. Například seskupení snímků podle kamery, ze které byly snímané, podle typu vad apod. Poslední částí je volba toho, jestli mají být na výstupu pouze anotace anebo anotace a originální snímky. Pokud je potřeba zvolit variantu anotaci spolu s originálními snímky, tak je potřeba zaškrtnout *Export with images*. Pro export originálních snímků je určité omezení, které je potřeba objasnit. Když se vytváří projekt v Cognex DLS, tak se na začátku do projektu importují snímky. Projekt jako takový se ukládá na disk spolu s podadresáři. V jednom z těchto podadresářů se nachází složka s veškerými importovanými snímky. V poskytnutých .NET knihovně není možný přímý přístup ke složce, ve které jsou uloženy tyto originální snímky, ale pouze k názvům snímků. Z těchto důvodu je nutné manuálně nastavit složku ve vytvořeném projektu, ve které jsou uloženy originální snímky. Nastavení složky se provádí přes parametr *Images directory* (Obr. 4.6). Po nastavení všech parametrů je možné zvolit tlačítko *Export*. Průběh export je možné sledovat přes ukazatel průběhu *Progress*.



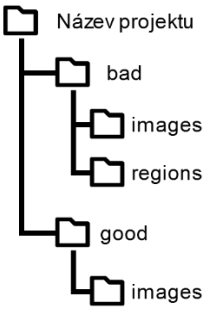
Obr. 4.6 *Plugin pro export anotací*

Na Obr. 4.7 je ukázka hlavní třídy, která se stará o propojení mezi Cognex DLS a samotným programem pro export anotací. Třída obsahuje celkem tři metody. První metoda IPlugin.Initialize se spouští na začátku a stará se hlavně o získání kontextu. Kontext představuje informace o právě otevřeném projektu. Po inicializaci se spustí metoda Run, která se postará o spuštění programu s grafickým rozhraním (Obr. 4.6).



Obr. 4.7 *Hlavní třída ExportLabelsPlugin*

Na Obr. 4.8 je ukázka struktury adresářů ve kterých jsou uloženy exportované originální snímky a anotace. Hlavní složka je označená názvem projektu v Cognex DLS. V této hlavní složce jsou složky *good* a *bad*. V Cognex DLS se využívají pro NOK snímky označení *bad* a pro OK snímek zase *good*. Ve složce *good* se nachází podadresář *images*, kde se uloží originální snímky, pokud je tato varianta zvolená. Když není zvolen export originálních snímků, tak se tyto složky *images* nevytvoří. A nakonec ve složce *bad* jsou dva podadresáře *images* a *regions*. Ve složce *regions* se ukládají samotné anotace. Anotace se ukládají jako binární obrázek ve formátu PNG. Anotace se ukládají pouze pro NOK snímky. Hodnoty pixelů jsou pro pozadí 0 a pro anotovaný defekt 255. Anotační obrázky, nebo také častěji segmentační mapy, mají stejné rozlišení i název jako je příslušný originální snímek, tak aby bylo možné spárovat anotace s originálními snímky.



Obr. 4.8 *Ukázka struktury adresářů exportovaných dat*

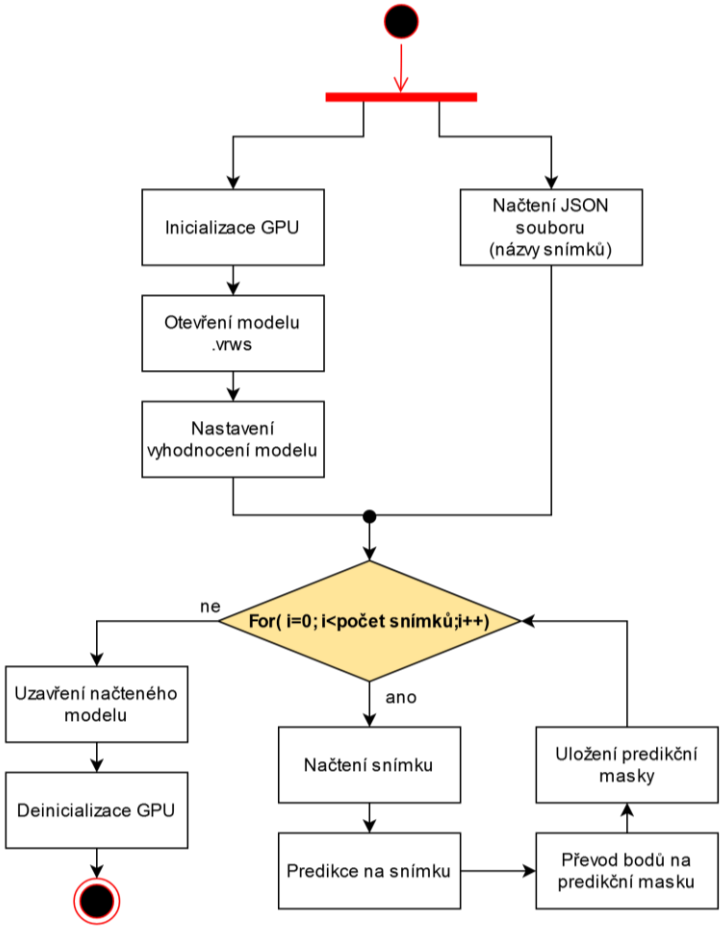
4.3.3 Evaluace Cognex modelu

Po natrénování modelu v Cognex DLS popsaného v 4.3.1, bylo nutné provést exportovat tzv. runtime modelu ve formátu VRWS. Cognex VisionPro Deep Learning poskytuje integraci jejich runtime modelu pomocí knihoven jazyka C anebo knihovny pro .NET. Pro využití .NET knihoven je možné je doinstalovat knihovny do Visual Studia pomocí NuGet packages, které se nachází v podadresáři aplikace Cognex DLS. Zvolená byla varianta .NET knihoven a jazyka C#. Pomocí těchto knihoven byly vytvořeny funkce, které byly zaintegrovány do funkcí v LabVIEW.

Na Obr. 4.9 je diagram vytvořené LabVIEW aplikace pro evaluaci Cognex modelu. Na začátku se provede inicializace zařízení, na kterém se bude provádět evaluace. Je možnost volby mezi grafickou

kartou a procesorem. Při nejčastější volbě grafické karty, je dále u ní možné nastavit velikost alokované paměti. Po inicializaci se provede načtení modelu ze souboru s příponou VRWS. Model se načte do operační paměti počítače anebo do paměti grafické karty. Načtený model je možný dále nastavovat. Je možné nastavit Feature size, Sampling density a Threshold (bližší popis 3.1.1). Pokud se nepovolí nastavení parametrů, tak se využijí parametry, se kterými byl model exportovaný z Cognex DLS. Kromě už napsaných parametrů, je možné nastavit minimální velikost hledaného defektu. Poslední parametr umožňuje zvolit typ výsledku. Na výběr je View, View+Region (without outer) a View+Region. První volba představuje výsledek za celý snímek v podobě klasifikačního skóre. Druhý výběr přidává na výstup pole clusterů, kde jeden prvek popisuje jeden nalezený defekt, a to parametry typu plocha defektu, skóre defektu apod. Poslední výběr přidává do clusteru pole souřadnic ohraničující nalezený defekt.

Paralelně s inicializací a načtením modelu, se provede načtení JSON souboru. V tomto souboru je uložený seznam indexů a názvů snímků. Indexy jsou využity jako jedinečný identifikátor pro práci s vlastní databází v testovací aplikaci popsané v 4.6.1.

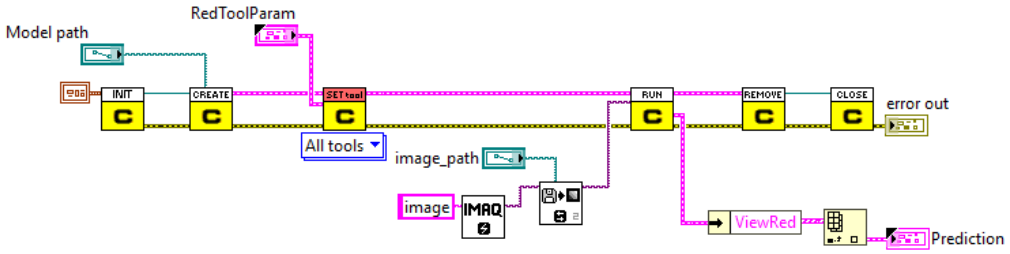


Obr. 4.9 *Diagram evaluace Cognex modelu*

Po načtení modelu a JSON souboru je možné začít v LabVIEW aplikaci postupně načítat snímky. Každý načtený snímek se převede na 2D pole a potom se převede 2D pole na 1D pole, které se přes Invoke Method přenese do vytvořené C# metody. Spolu s 1D polem se přenáší i informace o velikosti snímku a počtu kanálů pro převod zpět na 2D pole. Důvodem proč se převádí pole na 1D je, protože je

přenos přes LabVIEW Invoke Method pro 2D pole značně pomalý. Rozdíl může být až deseti násobný, ale to se odvíjí od velikosti 2D pole.

Jakmile se přenese snímek a vyhodnotí se na načteném modelu, tak se výsledky zapíšou do struktury pošlou se v podobě návratové hodnoty. Na straně LabVIEW se struktura vrací v podobě reference. Pro přístup k položkám struktury je potřeba využít .NET Property Node.



Obr. 4.10 *Ukázka vyhodnocení jedno snímku na Cognex modelu v LabVIEW*

Pro další zpracování je nutné mít predikční mapy ve formátu PNG obrázků. Jelikož je jediná volba výsledku v podobě souřadnic ohraničující nalezené defekty, tak bylo nutné tyto souřadnice převést na obrázky. Postup byl na začátku ve vytvoření prázdného obrázku (všechny pixely nabývaly hodnoty 0). Vytvořený obrázek má stejnou velikost jako snímek, na kterém se prováděla predikce. Následně se vzaly souřadnice defektů a vložily se do prázdného snímku pomocí funkce IMAQ Overlay Lines 2. Funkce umožňuje nastavit výplň vloženého objektu, konkrétně byla nastavena na 255. Jelikož se tímto vložily objekty pouze jako nedestruktivní vrstva, tak bylo potřeba ještě využít funkci IMAQ Merge Overlay, která zajistí destruktivní zapsání vrstvy do obrázku.

Vytvořena predikční mapa se následně uloží. Zároveň už bylo zmíněné, že model na výstupu poskytuje klasifikační skóre, které je důležité pro další zpracování, proto bylo spolu s indexem z JSON souboru spárovanou v podobě clusteru. Po vyhodnocení všech snímků vzniklo pole těchto clusterů s výsledky, a proto byly uloženy do JSON souboru. Tento soubor se následně otevírá v testovací aplikaci (4.6).

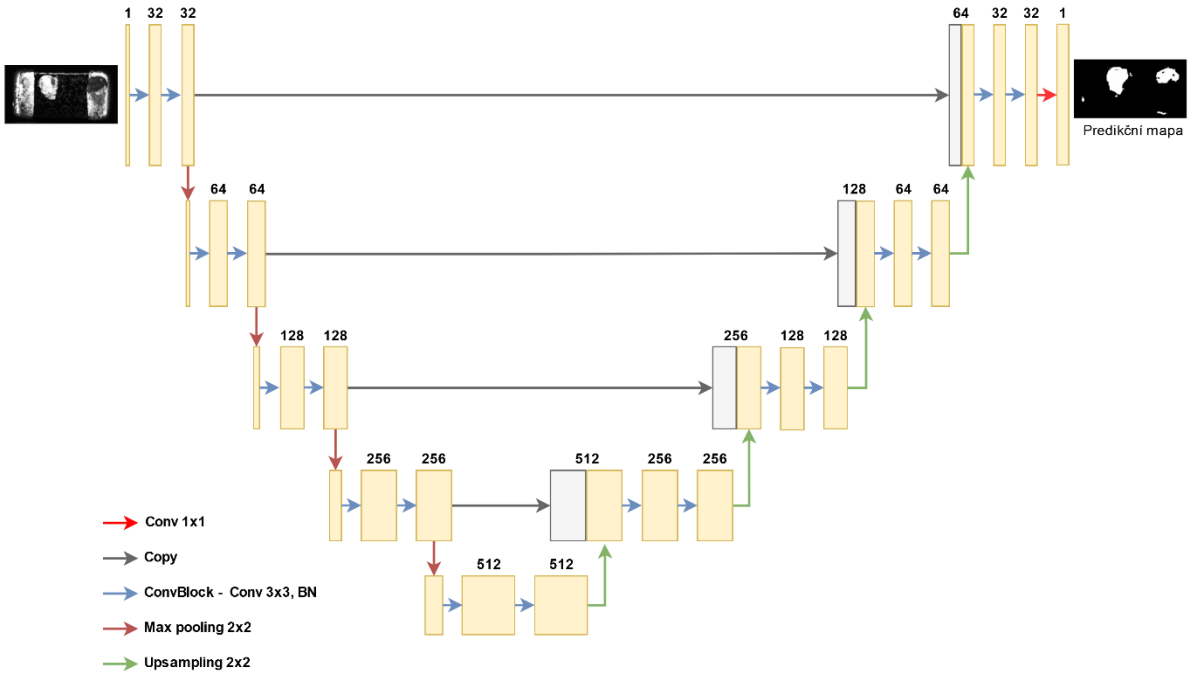
4.4 Architektury neuronových sítí

V této diplomové práci už byl popsaný modelu od společnosti Cognex v podkapitole 4.3.1, jež se jedná o komerční architekturu, která není veřejně přístupná. Pro diplomovou práci, ale byly využity další dvě nekomerční architektury, které jsou veřejně dostupné. Konkrétně jsou využity architektury U-Net [6] a SegDecNet [7]. Architektury jsou implementovány v jazyce Python pomocí knihovny TensorFlow a Keras. V podkapitolách 4.4.1 a 4.4.2 jsou blíže popsány použité architektury a v podkapitole 4.5 je popsané trénovaní implementovaných architektur.

4.4.1 U-Net

Jedná se o architekturu konvoluční neuronové sítě pro sémantickou segmentaci. Je to architektura byla původně vyvinutá pro vyhodnocení snímku z oblasti biomedicíny. Skládá se ze dvou částí: kontrakční (také nazývané enkodér) a expanzní (také nazývané dekodér). Na Obr. 4.11 je ukázka použité architektury U-Net, šipky znázorňují konkrétní transformaci v podobě vrstvy, žluté bloky

představují výstupní počet kanálů 3D tenzoru (počet filtrů v případě konvoluční vrstvy). Kontrakční část je typickou architekturou pro konvoluční sítě, která se skládá z aplikace 2D konvoluce s velikostí jádra 3x3, aktivační funkcí ReLU a definovaným počtem filtrů. Po dvou vrstvách 2D konvoluce následuje vrstva Max pooling 2x2 s krokem 2 (sdružovaní dle maxima viz podkapitola 2.2.1) (tmavě červená šipka Obr. 4.11). Vrstvou Max pooling se provede převzorkování, které pokaždé zdvojnásobí počet kanálů s příznaky (features). Druhá část expanzní je založena na vrstvě Upsampling 2x2 (zelená šipka Obr. 4.11), které je opakem vrstvy Max pooling. Vrstva Upsampling sníží počet kanálu na polovinu, po snížení se připojí k příslušné vrstvě z kontrakční části (šedý blok Obr. 4.11). Po připojení se opakuje dvakrát vrstva 2D konvoluce s velikostí jádra 3x3 a aktivační funkcí ReLU. Poslední finální vrstva je 2D konvoluce s jádrem 1x1 a aktivační funkcí Sigmoid, která složí k namapováni výstupní predikční mapy. Výstupní predikční mapa je ve velikosti vstupního snímku a hodnota pixelů se pohybuje v množině $p = \{0, 1\}$. [6]



Obr. 4.11 *Modifikovaná architektura U-Net*

Na Obr. 4.11 je ukázaná modifikovaná architektura U-Net, není původní architekturou, ale pro účely diplomové práce byla mírně upravená. Konkrétně za každou vrstvou 2D konvoluce 3x3, byla doplněna vrstvou dávkové normalizace (BN – Batch Normalization 2.2.3). Poslední rozdíl je, že originální U-Net začíná u 2D konvoluce s filtrem 64, ale v této práci byl změněn filtr na 32. Snížením filtru dojde k celkovém snížení počtu parametrů sítě a tím se sníží pravděpodobnost přetrénování (overfitting) modelu.

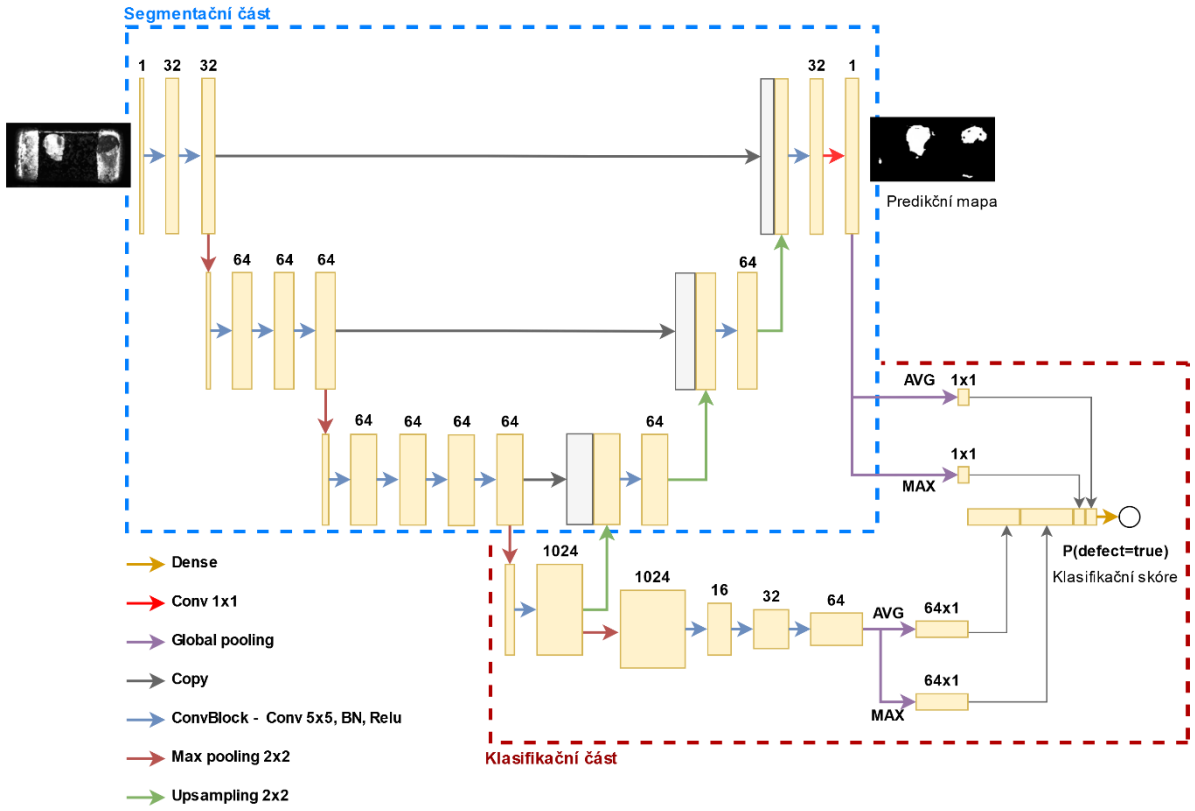
4.4.2 SegDecNet

Jedná se architekturu, která kombinuje architekturu pro segmentaci a pro klasifikaci. Architektura má dva výstupy, první je predikční mapa (jako U-Net) a druhým výstupem je klasifikační skóre, které udává pravděpodobnost, že se jedná o defektní snímek. Architektura byla publikovaná v článku pod názvem *Segmentation-based deep-learning approach for surface-defect detection* [7].

Z názvu už vyplívá, že se jedná o architekturu, která byla vyvinuta s cílem detekce povrchových defektů na průmyslových výrobcích.

Na Obr. 4.12 je ukázka modifikované architektury SegDecNet. Rozdíl oproti původní architektuře [7] je v doplnění expanzí části u segmentační části a díky tomu je výstupní predikční mapa ve stejné velikosti jako je vstupní snímek. Původní architektura měla na výstupu zmenšenou predikční mapu. Segmentační část je obdobná už popsané architektuře U-Net s tím rozdílem, že se v případě vrstvy 2D konvoluce se používá jádro 5x5.

Klasifikační část je řešená připojením na poslední vrstvu kontrakční části segmentační části. Jedna z klíčových prvků klasifikační části, je použití výsledků z finální predikční mapy pro získání klasifikační skóre, tímto propojením vznikne zkratka, která zajistí lepší konvergenci i při nižším počtu parametrů dané architektury. Na propojení se využívá globální maximální a průměrné sdružení (viz podkapitola 2.2.2).



Obr. 4.12 *Modifikovaná architektura SegDecNet*

4.5 Program pro trénovaní neuronových sítí

Po implementaci architektur z podkapitol 4.4.1 a 4.4.2 bylo dalším krokem vytvořit program pro trénování modelů pomocí knihovny TensorFlow. Program je bez grafického rozhraní, a proto úpravy trénovacích parametrů se provádí přímou editací proměnných v trénovacím programu. Pro vývoj programu byl využit jazyk Python ve verzi 3.6 a vývojové prostředí Pycharm 2020 Community edition.

Trénovaní neuronových sítí je tvořeno několika kroky: nahrání trénovacích dat, předzpracovaní dat, trénování modelu, a nakonec uložení natrénovaného modelu. V následujících podkapitolách jsou rozvedeny jednotlivé kroky.

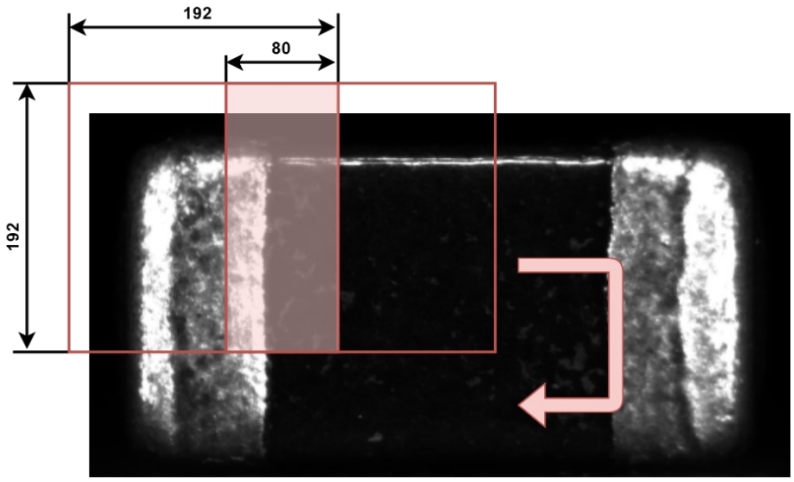
4.5.1 Příprava trénovacích dat

Při trénování neuronových sítí je nutné dodávat data, na které se bude neuronová síť učit a také cílové hodnoty podle kterých se počítá ztrátová funkce. Pro tyto účely je potřeba vytvořit generátor dat, který je možné iterovat pro získaní dat. V rozboru nástrojů byl popsaný skupina funkcí tf.data.Dataset použitých pro vytvoření generátoru dat (podkapitola 3.2.1). Funkce tf.data.Dataset byly použity pro vytvoření celkem tří různých generátorů dat.

Před popisem samotného generátoru a jeho variant je potřeba vysvětlit dvě metody úpravy snímků před vstupem do neuronové sítě. První metoda je využít celý snímek pro trénování. Druhá metoda je rozdělení snímku na menší části, na kterých se následně trénuje. Jednotlivé menší části jsou označované jako tzv. **patch** a metoda je potom nazývaná jako **patch-based**. Metoda patch-based se hlavně využívá pro trénování snímků s vysokým rozlišením a není tedy pro ně možné alokovat dostatek paměti na grafické kartě. Přesto je metoda patch-based výhodná i pro menší snímky, protože zlepšuje konvergenci modelu. Konvergence se zvýší, protože se rozdělením snímku na menší části uměle zvedne počet trénovacích dat. Pro diplomovou práci byly využity obě metody jak metoda využívající celý snímek, tak patch-based metoda. V kapitole 5 je srovnání těchto metod na architektuře U-Net. Pro architekturu SegDecNet byla využita pouze metoda využívající celé snímky. V následujících odstavcích jsou blíže rozvedeny obě metody.

Neuronové sítě pro segmentaci nemůžou na vstupu přijmou snímek v libovolném rozlišení. Pravidlem pro rozlišení vstupního snímku je dělitelnost šířky, a i výšky číslem 32. Použitý dataset SMD součástek nesplňuje tuto podmínku, protože výška se pohybuje mezi 250 až 262 pixely a šířka snímku je v rozmezí 495 až 510 pixelů. Řešením tohoto problému je doplnění nulami (tzv. padding) do rozlišení, které je dělitelné 32. V případě datasetu SMD součástek se jedná rozlišení 288x512. Tento postup doplnění nulami je využitý u metody využívající celé snímky pro trénování.

Druhá metoda patch-based spočívá v rozdělení snímku na menší části (patches). Na Obr. 4.13 je ukázka toho, jak probíhá rozdělení snímku. Důležitými parametry jsou okno a překryv. Okno představuje velikost vyříznutého snímku. Šířka a výška okna musí být dělitelná číslem 32. Dále překryv udává, kolik pixelů má být překryv s předešlým oknem. Na Obr. 4.13 je okno ve velikost 192x192 s překryvem oken 80 pixelů. Části oken, které jsou mimo plochu snímku jsou obdobně jako u první metody doplněny nulami.



Obr. 4.13 *Rozdělení snímku na menší části (patches)*

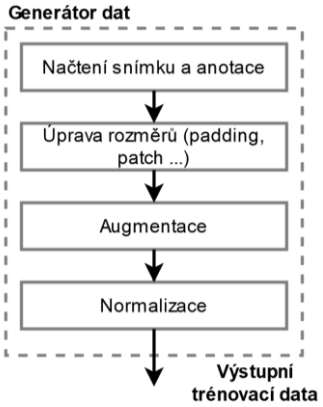
Pro metodu patch-based se provede výpočet počtu kroků jak ve směru šířky, tak ve směru výšky, pomocí vzorců (4.8). Celkový počet vyříznutých oken je potom vynásobení $step_w$ a $step_h$. Pro případ ukázaný na Obr. 4.13 je to celkem 8 snímků s rozlišením 192x192. Výpočet počtu sloupců a řádků, které je nutné doplnit nulami (padding), je popsaný vzorci (4.9).

w … $šířka\ snímku$
h … $výška\ snímku$
o . . $velikost\ okna$
p … $překryv$

$$step_w = \frac{w - o_w}{o_w - p} + 1\ , \quad step_h = \frac{h - o_h}{o_h - p} + 1 \tag{4.8}$$

$$padding_w = step_w \cdot (o_w - p) - (w - o_w)\ , \quad padding_h = step_h \cdot (o_h - p) - (h - o_h) \tag{4.9}$$

Soubor funkcí tf.data.Dataset nedefinuje konkrétní operace pro načtení a ani předzpracování dat. Funkce slouží pouze pro vytvoření sekvence operací, ale jednotlivé operace jako načtení dat z disku, škálování obrazových dat, augmentace apod., je nutné doimplementovat. Implementované operace jsou ukázané na Obr. 4.14. Prvním krokem je načtení snímku a segmentační mapy uložené v PNG formátu. Následně se provedla úprava snímků podle zvolené metody. V případě první metody operující s celým snímkem se provedlo doplnění rozměrů snímku i anotace, tak aby šířka i délka byly dělitelné číslem 32. U druhé metody patch-based se provedl na začátku výpočet počtu kroků (4.8) a počet řádků a sloupců (4.9), které je potřeba doplnit nulami. Potom se mohla spustit WHILE smyčka, ve která se postupné extrahovaly patches a v případě okna u okrajů snímku se provádělo doplnění nulami. Tato operace se provedla pro snímek a anotaci.



Obr. 4.14 *Sekvence operací v generátoru dat*

Před posledním krokem (Obr. 4.14) byla aplikace augmentace. K provedení augmentace se využila knihovna Albumentations (podkapitola 3.4). Poslední krok byla normalizace snímku, a to podělením pixelů hodnotou 255. Normalizace se provádí z důvodů, že není vhodné používat pro učení neuronových sítí poměrně vysoká celá čísla, protože jsou výrazně vyšší než počáteční hodnoty sítě, a to může vést k vypnutí rozsáhlých aktualizací gradientů, které zabrání konvergenci sítě [2]. Potřebné je normalizovat i anotace.

V Tab. 4.7 jsou popsány jaké jsou výstupy z vytvořených třech typů generátorů. Všechny výstupy v Tab. 4.7 představují jeden datový bod, ale ve skutečnosti z generátoru vychází dávka (batch) těchto datových bodů. První typ představuje na výstupu základní konfiguraci snímku spolu se segmentační mapou. Druhý má na výstupu je patch snímku a segmentační mapy. A poslední typ má na výstupu snímek spolu se segmentační mapou a cílová klasifikační třída ($třída \in [0; 1]$).

Tab. 4.7 *Výstup z generátoru*

Generátor	Typ sítě	Výstupní trénovací data
1	U-Net	Snímek, Segmentační mapa
2	U-Net	Patch-Snímek, Patch-Segmentační mapa
3	SegDecNet	Snímek, (Segmentační mapa, Klasifikační třída)

4.5.2 Trénování modelu

Po vytvoření generátoru popsaného v 4.5.1 je možné přistoupit k trénování modelu. Před samotným zahájením je nutné provést kompilaci implementovaných architektur neuronových sítí.

Pro kompilaci modelu je potřeba zvolit ztrátovou funkci, optimalizátor a metriky. Pro trénování byly zvoleny celkem dvě ztrátové funkce. První je ztrátovou funkcí je křížová entropie, která je počítaná mezi cílovou hodnotou p a predikovanou hodnotou q. Rovnice (4.10) je ukázka obecného výpočtu křížové entropie, kde i je index každého elementu cílové a predikované hodnoty. [9]

$$H(p, q) = -\sum_{i} p_i \cdot \log(q_i) \tag{4.10}$$

Jak už bylo několikrát zmíněno, tak se v této práci výhradně operuje s binární klasifikací a segmentací. A tudíž je možné zjednodušit výpočet křížové entropie, protože cílová hodnota nabývá pouze dvou hodnot $y = 1$ a $y = 0$, tak potom $p\epsilon\{y; 1 - y\}$ a $q\epsilon\{\hat{y}; 1 - \hat{y}\}$ a výsledný zjednodušený vzorec je (4.11). [9]

$$H(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})) \tag{4.11}$$

Finální ztrátová funkce je vzorec (4.12), kde m je iterace přes trénovací data a cílem je minimalizovat tuto hodnotu. Funkce je definovaná pro klasifikační úlohu. Pro segmentační úlohy je rozdíl, že se křížová entropie počítá pro každý pixel. [9]

$$L_{BCE}(y, \hat{y}) = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)) \tag{4.12}$$

Druhou použitou ztrátovou funkcí je Tversky. Jedná se o ztrátovou funkci používanou pro sémantickou segmentaci. Funkce vychází z Tversky indexu, který je ukázaný ve vzorci (4.13). Jedná se o zobecnění evaluační metriky F1 popsané v podkapitole 4.2.3, kde je navíc přidané váhy pro FP a FN. Při nastavení vah α a β na hodnotu 0,5 se získá metrika F1. Váhy by se měli vždy volit, tak aby suma $\alpha + \beta = 1$. Ve vzorci (4.13) je p pro segmentační mapu a $\hat{p}$ je pro predikční mapu. [8]

$$TI\ (p, \hat{p}) = \frac{p \cdot \hat{p}}{p \cdot \hat{p} + \alpha \cdot (1 - p) \cdot \hat{p} + \beta \cdot p \cdot (1 - \hat{p})} \tag{4.13}$$

Ztrátová funkce Tversky je ukázaná ve vzorci (4.14) a ve vzorci (4.15) je definovaná ztrátová funkce Tversky vzhledem k hodnotám TP, FN a FP. Ze vzorce je (4.15) je zřejmé, že váhami α a β je možné kompenzovat nevyváženost datasetu, která je častým problémem v oblasti sémantické segmentace. Nastavením $\alpha < \beta$ se přidá větší důležitost FN výsledkům a to znamená, že učení sítě se zaměří na detekci hledané třídy oproti pozadí, které je ve snímcích často zastoupeno ve větší míře. [8]

$$L_{Tversky} = 1 - TI \tag{4.14}$$

$$L_{Tversky} = 1 - \frac{TP}{TP + \beta \cdot FN + \alpha \cdot FP} \tag{4.15}$$

Další parametr kompilace je optimalizátor, který implementuje stochastický gradientní sestup. Zvolený byl optimalizátor Adam, který se dovede adaptivně přizpůsobit během trénování tzv. rychlost učení (dále learning rate). Learning rate představuje parametr, který říká optimalizátoru, jak rychle minimalizovat ztrátovou funkci. Když je learning rate příliš vysoká, tak optimalizátor má problém konvergovat k minimu. Naopak pokud je příliš nízká, tak se výrazně zpomalí konvergence anebo může se dostat špatného lokálního minima. [9]

Poslední parametr kompilace jsou metriky. Pro trénování byla zvolena pouze jedna metrika, a to konkrétně metrika F1 (4.2.3). Metrika je počítaná po každé dokončené epoše a umožňuje, tak průběžně sledovat kvalitu trénovaného modelu.

Po dokončení kompilace je možné začít trénování modelu. Pro trénování je potřeba nastavit počet iterací. Výpočet se liší podle metody použité v generátoru dat (4.5.1). V případě metody využívající celé snímky je výpočet podle vzorce (4.16), který je počítaný jako dávka (batch) vynásobená počtem zvolený epoch. Pro druhou metodu patch based je výpočet podle vzorce (4.17), který je doplněný hodnotou počtu kroků (počet patches).

$$iterations = batch \cdot epochs \tag{4.16}$$

$$iterations = batch \cdot (step_w \cdot step_h) \cdot epochs \tag{4.17}$$

4.5.3 Nastavení trénovacích parametrů

V této podkapitole je uvedeno, jak jsou nastavené parametry pro trénování jednotlivých architektur. Byly provedeny celkem čtyři varianty testů, které kombinují implementované architektury a metody popsané v předešlých podkapitolách. V Tab. 4.8 jsou popsané varianty modelů, jejichž výsledky jsou srovnány v kapitole 5. První dvě testované varianty představují architekturu U-Net a s využitou metodou patch-based. Rozdíl spočívá s rozdílným nastavením okna a tím i odpovídající velikostí vstupu modelů. Velikost překryvu oken je nastavená v obou případech na 80 pixelů. Třetí variantou je architektura U-Net s metodou využívající celé snímky. Poslední varianta je architektura SegDecNet, která je stejně jako předešlá, trénovaná s metodou využívající celé snímky. Velikost dávky (batch size) se liší vzhledem k velikosti trénovaných snímků. Menší snímky alokují méně paměti na grafické kartě a tím pádem je možné nastavit větší dávku těchto snímků.

Tab. 4.8 *Navržené test pro srovnání odlišných přístupů*

T.č.	Architektura	Metoda	Velikost vstupu	Překryv (px)	Velikost dávky
1	U-Net	Patch based	128x128	80	60
2	U-Net	Patch based	192x192	80	45
3	U-Net	Celý snímek	288x512	-	11
4	SegDecNet	Celý snímek	288x512	-	11

Jednotlivé testy popsané v Tab. 4.8 sdílejí také společné trénovací parametry popsané v Tab. 4.9. Důležité je se zmínit o ztrátové funkci Tversky, která umožňuje nastavit parametry α a β (4.5.2). konkrétně byly na základě experimentování zvolené parametry $\alpha = 0{,}3$ a $\beta = 0{,}7$, které kladou větší důležitost na FN výsledky. Tento přístup je zvláště důležitý v kontextu hledání defektů, protože je důležitější snížit počet nepredikovaných defektů (FP) než falešně predikovaných defektů (FN).

Tab. 4.9 *Společné trénovací parametry*

Počet epoch	40
Ztrátová funkce - Tversky	$\alpha = 0{,}3 \quad \beta = 0{,}7$
Learning rate	0,001
Optimalizátor	Adam

V podkapitole 4.5.2 byly teoreticky popsány použité ztrátové funkce. Vzorec (4.18) představuje použitou finální ztrátovou funkci pro segmentaci. Je kombinací binární křížové entropie L_{BCE} a funkcí Tversky $L_{Tversky}$. Tato ztrátová funkce byla použita pro trénování všech variant popsaných v Tab. 4.8.

$$L_{segm} = L_{BCE} + L_{Tversky} \tag{4.18}$$

Jak už bylo popsané v podkapitole 4.4.2, tak architektura SegDecNet má na výstupu predikční mapu, která se použije spolu se segmentační mapu pro výpočet ztrátové funkce L_{segm}. Zároveň má SegDecNet na výstupu klasifikační skóre. Pro výpočet ztrátové funkce z klasifikačního skóre a cílové třídy byla využitá binární křížová entropie L_{BCE} definovaná ve vzorci (4.19). V případě kompilace modelu SegDecNet bylo tedy potřeba zadat dvě ztrátové funkce. První funkce je pro segmentaci L_{segm} a druhá pro klasifikaci L_{class}. V rámci kompilace je zároveň možné zvolit váhu pro každou ztrátovou funkci. Konkrétně bylo zvolené snížení důležitosti ztrátové funkce L_{class} na polovinu. Důležitost ztrátová funkce L_{segm} nebyla snižovaná. Důvodem snížení klasifikační části je, protože by při trénování negativně ovlivnila trénování segmentace.

$$L_{class} = L_{BCE} \tag{4.19}$$

V podkapitole 4.5.2 byl popsaný optimalizátor, který zvládne adaptivně přizpůsobit learning rate. Parametr learning rate je potřeba v průběhu trénovaní postupně snižovat. Výchozí hodnota je v Tab. 4.9. Adaptivní nastavení není dostačující, a proto se využila už v TensorFlow vytvořená callback funkce LearningRateScheduler, které se automaticky volá na konci dokončené epochy. Do této callback funkce je potřeba definovat vlastní podmínky a funkce na základě kterých se provede snížení learning rate. Pro trénování bylo nastavený pokles learning rate od desáté epochy, a to podle funkce (4.20). Ve funkci lr_{prev} prezentuje hodnotu learning rate v předešlé epoše a lr_{new} prezentuje novou hodnotu learning rate. Pokles je počítaný z přirozené exponenciální funkce e^{-x}, kde velikost exponentu $-x$ definuje míru změny. Experimentálně byla nalezena hodnota exponentu $-0{,}1$.

$$lr_{new} = lr_{prev} \cdot e^{-0{,}1} \tag{4.20}$$

V Tab. 4.10 je ukázka zvolené augmentace snímků a anotací. Při běhu augmentační sekvence je nežádoucí, aby se aplikovaly všechny na vyhodnoceném snímku. Důvodem je, že by se takovým přístupem prováděla, tak by se síť neučila vůbec na originálních snímcích, ale pouze silně augmentových snímcích. Knihovna Albumentatotions nabízí řešení tohoto problému pomocí seskupení několika transformací a při běhu se z dané skupiny vybere pouze jedna transformace. Zároveň knihovna Albumentations poskytuje nastavení u každé transformace pravděpodobnost provedení dané augmentace. Konkrétně byla u všech transformací nastavená pravděpodobnost $p = 0{,}3$. Použitá pravděpodobnost indikuje, že se jedná o lehčí augmentaci. Vyšší pravděpodobnost je vhodnější menší datasety snímků.

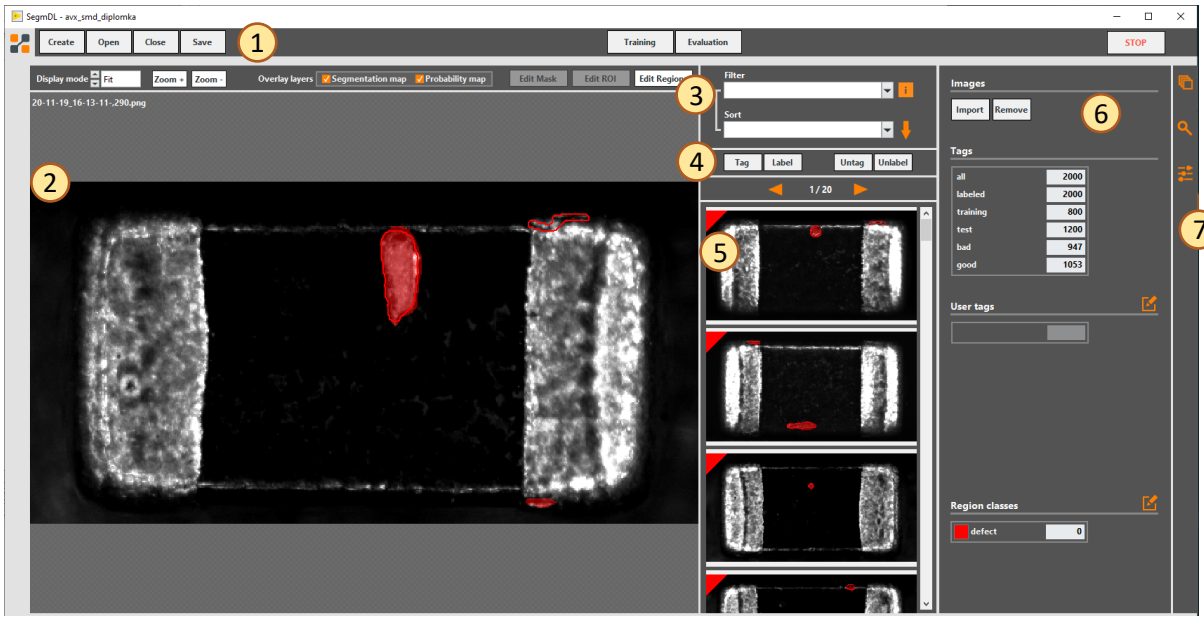
Zvolené augmentační transformace (Tab. 4.10) byly zvoleny, tak aby odrážely použitý dataset SMD součástek. Například v kontextu použitého datasetu by nedávalo smysl použít transformace jako náhodný vyříznutí okna ze snímku, elastickou deformaci apod. Jedna z použitých transformací je např. posun, který provede náhodné posunutí snímku ve směru x a y. Maximální velikost posunu je definovaná v procentech vzhledem k rozměrům snímku.

Tab. 4.10 *Použité augmentační transformace*

Transformace
Horizontální překlopení
Vertikální překlopení
Rotace [-1°; +1°]
Posun [-6%; +6%]
Ostření
Jas [-2%; +2%]
Kontrast [-2%; +2%]

4.6 Testovací software

Poslední částí praktické realizace bylo vytvořit aplikaci v LabVIEW, která bude určena pro evaluaci a vyhodnocení výsledků z natrénovaných modelů, které byly popsány v předešlých podkapitolách. Provádění evaluace není možné pomocí LabVIEW funkcí, ale je nutné využít metody vytvořené v jazyku Python s knihovnou TensorFlow. Vývojové prostředí LabVIEW poskytuje tzv. Python Node pro spouštění těchto funkcí (bližší popis 3.5.2). Na Obr. 4.15 je ukázka vytvořené aplikace.



Obr. 4.15 *Testovací aplikace*

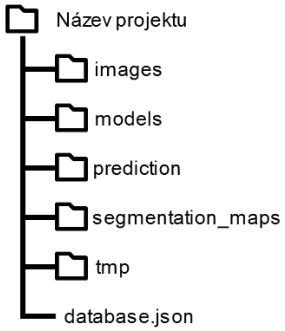
Hlavní části aplikace:

1. Pás nabídky – nástroje pro práci s projektem, evaluace, trénovaní a zastavení aplikace
2. Hlavní displej snímků
3. Nástroje pro filtraci a roztřídění snímků v datasetu
4. Nástroje pro označení (tagování) a anotovaní snímků v datasetu
5. Prohlížeč snímků
6. Okna pro dataset, evaluační metriky a nastavení

7. Lišta pro přepínání oken pro dataset, evaluační metriky a nastavení

Jádro aplikace je postaveno na stavovém automatu řízeného událostmi. Pro zvýšení přehlednosti a modularity jsou celkem dvě další aplikace spouštěné dynamicky. Konkrétně se jedná o VI hlavního displeje, který se spustí v subpanelu. Druhé dynamické VI, které se spustí jako samostatné okno, je určené pro evaluaci modelu (4.6.2). Komunikace mezi hlavním VI a dynamicky spuštěními VIs probíhá přes tzv. user events (uživatelské události), jejichž reference jsou vytvořeny jako globální proměnné. Globální proměnné podporují sdílení paměťového prostoru mezi nezávislými VIs. Vytvořené globální user events jsou odchyceny pomocí Event structure. User events umožňují přenášet data. V této aplikaci je nutné přenášet rozdílná data, proto bylo přistoupeno k použití datové struktury variant, jež umoňuje zapouzdřit libovolnou datovou strukturu (cluster, numeric atd.).

Vytvořená aplikace je založená na vytváření projektů. Pro práci s projekty jsou určeny tlačítka v pásu nabídky (bod 1 Obr. 4.15). Dostupné jsou tlačítka pro vytvoření prázdného projektu, otevření, uložení a uzavření projektu. Při vytváření projektu musí uživatel zvolit název projektu, který se promítne do názvu složky projektu (Obr. 4.16) a zároveň se název aktuálně otevřeného projektu zobrazí v levém rohu hlavního okna aplikace. Struktura projektu (Obr. 4.16) je koncipovaná jako hlavní složka s názvem projektu. Podsložky projektu jsou *images*, *models*, *prediction*, *segmentation_maps* a *tmp*. Složka *images* je určena pro uložení veškerých importovaných snímků do projektu. Složka *models* je pro ukládání modelů, které jsou určeny pro evaluaci. Složka *prediction* je určená pro ukládání veškerých predikčních map ve formátu PNG. Složka *segmentations_maps* je obdobně jako *prediction*, určená pro uložení anotací ve formátu PNG obrázků. Poslední složka *tmp* je určená pro ukládání dočasných souborů nutných pro běh programu. V rámci složky projektu je přítomný i soubor *database.json*, který bude dále popsaný v podkapitole 4.6.1.

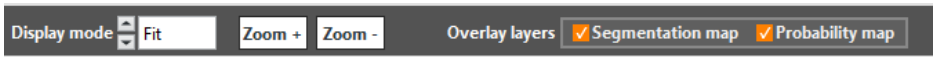


Obr. 4.16 *Struktura projektu*

V rámci pásu nabídky (bod 1 Obr. 4.15) jsou uprostřed tlačítka trénování a pro evaluaci. Tlačítko pro trénování provádí pouze export JSON souboru do složky *tmp*, který obsahuje seznam snímků a další důležité informace určené pro trénování v Python programu (podkapitola 4.5). Druhé tlačítko je určené pro spuštění aplikace určené pro evaluaci modelů, bližší popis této aplikace se nachází v podkapitole 4.6.2.

Další důležitou částí aplikace je hlavní displej snímků (bod 2 Obr. 4.15), který je určený pro zobrazení snímku. Kromě snímku se zobrazí i anotace snímků (segmentační mapa), která je znázorněná jako červený obrys kolem anotovaného defektu. Dále se zobrazí, pokud je dostupná predikční mapa (musí proběhnout evaluace modelu). Predikční mapa je znázorněná v podobě poloprůhledné červené mapy. Obě jak obrys anotace, tak i predikční mapa jsou vykresleny ve dvou nezávislých overlays.

Overlay je v LabVIEW terminologii vrstva, která umožňuje nedestruktivním způsobem zobrazit text, jiné obrazce apod. do zobrazovaného hlavního snímku. Pokud uživatel potřebuje zobrazit hlavní snímek, který zakrývají vrstvy, tak pro tyto účely jsou poskytnuty klávesové zkratky CTRL+levá šipka a CTRL+pravá šipka. Těmito klávesovými zkratkami je možné jednotlivé vrstvy skrýt anebo znovu ukázat. Skrývání a ukazování vrstev funguje obdobně jako obousměrný kruhový registr. Na Obr. 4.17 je ukázaný indikátor *Overlay layers*. Indikátor zobrazuje, které vrstvy jsou skryté, nebo zobrazené. Pokud například u OK snímku neexistuje anotace (segmentační mapa), tak je daný indikátor zašedlý. Dále se na nástroje liště (Obr. 4.17) nachází nástroje pro přiblížení snímku, nebo vyplnění displeje snímkem (fit).



Obr. 4.17 *Nástrojová lišta pro hlavní displej snímku*

Jak už bylo v předešlém odstavci zmíněno, tak anotace (segmentační mapa) jsou zobrazeny jako obrys defektu. Anotace jsou uloženy ve formátu PNG obrázků a bylo proto nutné převést je pomocí funkce IMAQ LabelToROI. Výstupem funkce je polygon, který je možný zobrazit jako overlay snímku. Pro predikční mapy bylo nutné provést matematické operace pro vytvoření transparentní mapy, která se zobrazila jako overlay.

Výpočet transparentní predikční mapy pro jeden pixel:

1. Pixel snímku je označen jako $pixel^{s}$ a pixel predikční mapy jako $pixel^{m}$.

$$pixel^{s} = (r^{s},\ g^{s}, b^{s}),\ \ pixel^{m} = (r^{m},\ g^{m}, b^{m})$$

2. Transparentnost predikční mapy je možné nastavit parametrem $t \in \langle 0, 1\rangle$, kde číslo 1 představuje 100% transparentnost, tedy bez predikční mapy

$$p = 1 - t$$

3. Potom se provede pře násobení koeficienty

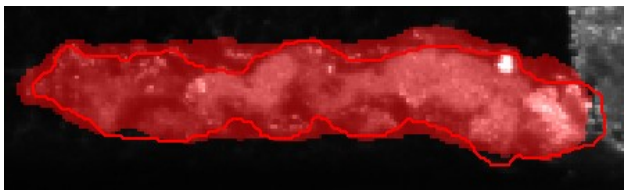
$$\hat{r}^{s} = t \cdot r^{s},\quad \hat{g}^{s} = t \cdot g^{s},\quad \hat{b}^{s} = t \cdot b^{s}$$

$$\hat{r}^{m} = p \cdot r^{m},\quad \hat{g}^{m} = p \cdot g^{m},\quad \hat{b}^{m} = p \cdot b^{m}$$

4. Nakonec se sečtou pixely a získá nový pixel $pixel^{f}$

$$pixel^{f} = \left(\hat{r}^{s} + \hat{r}^{m}, \hat{g}^{s} + \hat{g}^{m}, \hat{b}^{s} + \hat{b}^{m}\right)$$

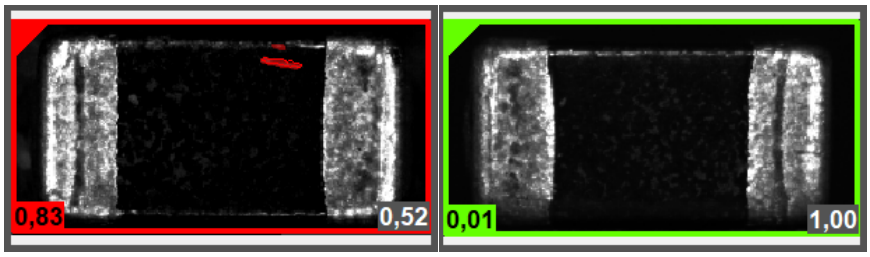
Jednotlivé operace je možné provádět postupně s každým pixel, nebo je možné využít funkce pro matematické a logické operace s obrázky. Konkrétně se jedná o knihovnu IMAQ Operators, která zajistí výrazně rychlejší zpracovaní než postupně pixel po pixelu. Snímky a segmentační mapy jsou odstínech šedi, a proto je pro zobrazení transparentní mapy potřebné převést na RGB formát. Na Obr. 4.18 je ukázka obrysu anotace (segmentační mapy) a transparentní predikční mapy.



Obr. 4.18 *Ukázka predikční mapy a anotace ve snímku*

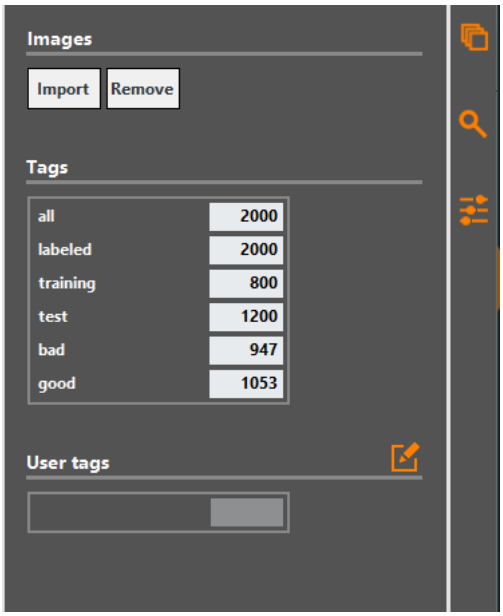
Další důležitou částí aplikace je prohlížeč snímků (bod 5 Obr. 4.15). Jedná se o v podstatě o pole snímků, která prošly filtrací. Filtrace je blíže popsaná v podkapitole 4.6.1. Vzhledem k tomu, že v datasetech se operuje v jednotkách tisíc snímků, tak by takové množství snímků bylo velice paměťově a časově náročné zobrazit najednou, z toho důvodu bylo přistoupeno k zobrazení pouze 100 snímků s možností se přepínat mezi stránkami. Počet stran se odvíjí od počtu snímků, které prošly filtrací. Přepínání stran je možné šipkami doleva a doprava ve formě tlačítek nad prohlížečem snímků.

Snímky, které se ukazují v prohlížeči snímků jsou zmenšeny pro snížení zatížení operační paměti. Konkrétně je největší strana snímku zmenšena na rozlišení 256 a druhá strana je zmenšena vzhledem k zachování poměru stran. U každého snímku je zobrazena třída OK a NOK a to formou trojúhelníku v levém horním rohu snímku (Obr. 4.19). V celé aplikace je konzistentně označovaná NOK třída červenou barvou a OK třída zelenou barvou. Dále každý snímek má v levém dolním rohu zobrazené klasifikační skóre a v pravém dolním rohuje je zobrazená hodnota IoU (viz 4.2.3).



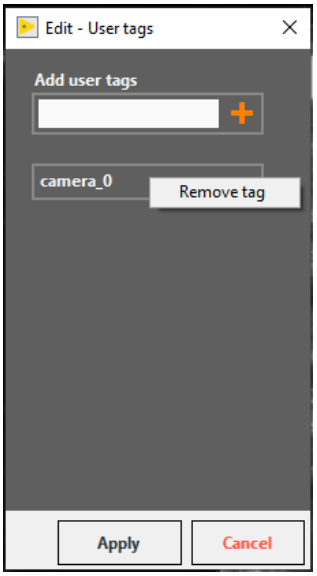
Obr. 4.19 *Ukázka dvou prvků z prohlížeče snímků*

Další částí aplikace je lišta s tlačítky (bod 7 Obr. 4.15). pro přepínání oken (bod 6 Obr. 4.15). konkrétně jsou dostupná tři okna: pro správu datasetu, evaluační metriky a nastavení. Okno pro nastavení zde není popsané, protože obsahuje několik parametrů pro ladění aplikace a nejsou příliš zásadní pro pochopení vytvořené aplikace. Další okno pro evaluační metriky je blíže popsané v podkapitole 4.6.3. Poslední okno pro správu datasetu je ukázané na Obr. 4.20. První částí jsou tlačítka pro import a smazání snímků z datasetu. V případě importu snímků se při kliknutí na tlačítko otevře vyskakovací okno, kde se zvolí složka se snímky, které se mají importovat. Zároveň je možné se snímky importovat anotace, u kterých se očekává stejný název jako u importovaných snímků. Smazání snímků z datasetu se provádí na základě zvoleného tagu.



Obr. 4.20 *Okno pro správu datasetu*

V textu již bylo několikrát použito slovo tag, pro které je možným synonymem kategorie, nebo set. To znamená, že tag s jedinečným definovaným názvem, který vytváří skupinu snímků. V aplikaci je defaultně 10 tagů: *all*, *labeled*, *good* (OK), *bad* (NOK), *training*, *test*, *TP*, *TN*, *FP* a *FN*. Veškeré importované snímky do projektu mají tag *all*. Všechny snímky, které byly označeny jako trénovací mají tag *training*. Všechny snímky, které byly označeny buď jako OK, nebo NOK, tak jsou zároveň tagovány jako *labeled* atd. Počty snímků s nějakým přiřazeným tagem jsou zobrazeny v části Tags (Obr. 4.20). Je důležité zmínit, že jeden snímek může mít více tagů, ale jeden konkrétní snímek může být v tagu zastoupen pouze jednou. Kromě od defaultně definovaných tagů je možné, aby uživatel vytvořil vlastní tagy. Vytvořit nový je možné tlačítek pro editaci v pravé části User tags (Obr. 4.20). Po zvolení editace se zobrazí vyskakovací okno na Obr. 4.21. V okně je možné přidat nový tag, anebo pomocí pravého tlačítka je možné otevřít menu s možností smazat tag.

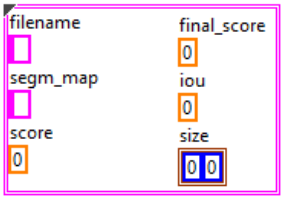


Obr. 4.21 *Editace uživatelský tagů*

4.6.1 Databáze

V rámci testovací aplikace je nutné operovat s velkým počtem snímků, které je potřeba třídit a seskupovat. Řešením tohoto problému bylo naprogramovat vlastní databázi, a to pomocí funkcí Set a Map v knihovně Collection (blíže popsaná v 3.5.3). Funkce Map jsou použity pro vytváření databáze snímků, kde jeden prvek mapy je zastoupený clusterem. Cluster (Obr. 4.22) obsahuje název snímku, název segmentační mapy (pokud neexistuje, tak obsahuje slovo „none“), klasifikační skóre získané z evaluace, finální skóre, hodnotu IoU, a nakonec rozlišení snímku. Každý snímek v mapě je mapován na přirozené číslo, které slouží jako jedinečný identifikátor v databázi. Funkce Set jsou použity pro tagy, kde jeden Set je určený pro jeden tag. V daném Setu jsou uloženy právě jedinečné identifikátory snímků.

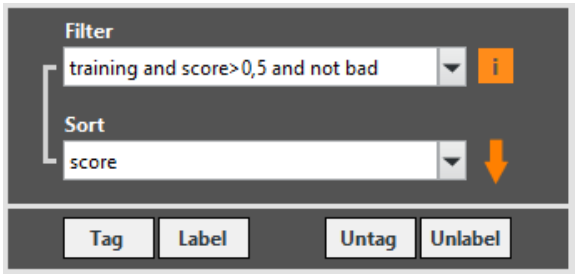
Databázi je nutné ukládat, a to ve formě souboru *database.json*, ve kterém jsou uloženy veškeré tagy a mapa snímků. Soubor se ukládá do hlavní složky projektu.



Obr. 4.22 *Cluster pro jeden snímek v mapě*

Pro práci s vytvořenou databází je nutné pochopit filtraci databáze. Na Obr. 4.23 je ukázka nástroje pro filtraci, respektive nejdůležitější je první control Filter. Jedná se v podstatě o příkazový řádek, kde na základě definovaných příkazu lze zvolit filtr. Filtrace je možná například na základě zadaného tagu. Po zvolení filtru se na pozadí provede filtrace databáze snímků, výstupem je Set, který obsahuje jedinečné identifikátory snímků. Tento vzniklý filtrovaný Set je dále nazývaný jako aktuální kontext databáze. Po dokončení filtrace a získání aktuální kontext databáze je možné aktualizovat zobrazené snímky v prohlížeči snímků.

Mimo už popsané jednoduché filtrace jedním tagem, je možné i zřetězit několik příkazů. Pro navázání více příkazu je dostupný logický operátor *and*. Současně je dostupný i logický operátor *not*, se který je možné negovat tag. Například pokud se zvolí příkaz *not training*, tak se zobrazí všechny snímky, které nejsou tagovány jako *training*. Pokud uživatel chce filtrovat podle uživatelsky vytvořeného tagu, tak je potřeba formovat příkaz podle *tag:název_tagu*. Posledním příkazem je možné omezit snímky podle jejich klasifikačního skóre, které nesplňují zadaný interval.

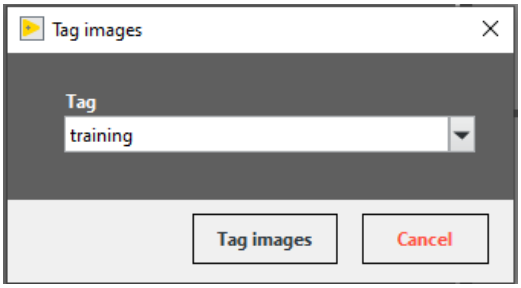


Obr. 4.23 *Nástroje pro filtraci a třídění databáze*

Kromě filtrace je možné provádět třídění zobrazených snímků v prohlížeči snímků. Třídit je možné pomocí kontrolu Sort, kde možné zvolit třídění buď podle klasifikačního skóre anebo podle hodnoty IoU. Třídění se vždy provádí pouze na aktuální kontext databáze. Je možné zároveň zvolit sestupný, nebo vzestupně třídění.

Pro názornost bude vysvětleny nastavený filtr na Obr. 4.23. Filtr omezí aktuální kontext databáze na snímky, které jsou tagované jako trénovací zároveň jejich klasifikační skóre je větší než 0,5 a zároveň nejsou tagovány jako NOK. Nakonec se provede třídění podle klasifikačního skóre a sestupně. Vzniklý aktuální kontext databáze se zobrazí v prohlížeči snímků.

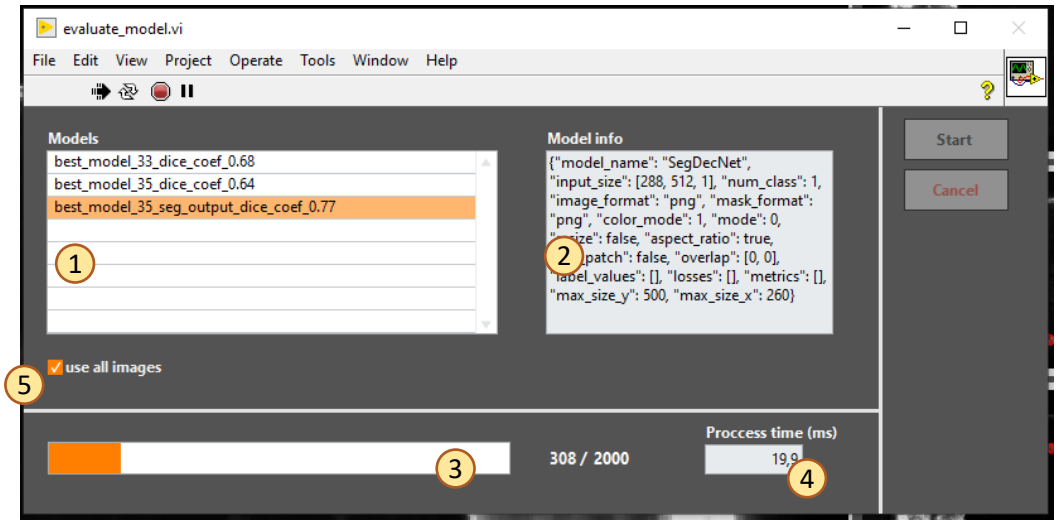
Pro práci s databází byla vytvořena další doplňující funkce. Na Obr. 4.23 jsou ukázané jako tlačítka *Tag*, *Label*, *Untag* a *Unlabel*. Jsou to funkce, které umožňují například tagovaní snímků, které se nachází v aktuálním kontext databáze. Na Obr. 4.24 je ukázané vyskakovací okno pro tagovaní, které nabízí volbu tagu ze seznamu. Je možné si všimnout, že jsou dostupné tlačítka *Label* a *Unlabel*, které složí pro přiřazení klasifikační třídy good (OK) a bad (NOK). Jedná se v kontextu databáze opět tagy, ale se speciální vlastností, a proto byly takto graficky odděleny. Rozdíl je, že pokud jeden konkrétní snímek může být pouze v jedné třídě OK, nebo NOK.



Obr. 4.24 *Vyskakovací okno pro tagovaní snímků*

4.6.2 Evaluace modelů

Pro evaluací modelu byla vytvořená samostatná aplikace, která se spouští dynamicky z hlavní aplikace. Na Obr. 4.25 je ukázka grafického rozhraní evaluační aplikace. V této evaluační aplikaci se přistupuje k vytvořenému Python kódu, který má importovanou knihovnu TensorFlow. V aplikaci se v seznamu (bod 1 Obr. 4.25) zobrazí veškeré dostupné modely uložené ve složce *models* (Obr. 4.16). Model natrénovaný v TensorFlow se ukládá ve formátu H5, zároveň s tímto model se provádí uložení souboru JSON s uloženými parametry modelu. Po zvolení modelu ze seznamu se zobrazí tyto parametry v indikátoru (bod 2 Obr. 4.25). V souboru jsou např uloženy informace typu modelu, velikosti vstupního snímku a další. Tento soubor je zcela zásadní pro opětovně spouštění modelu. Po zvolení model je dále možné zvolit, které snímky se mají použít pro vyhodnocení (bod 2 Obr. 4.25), na výběr jsou všechny v datasetu, anebo pouze testovací sada. Snímky, které se mají použít pro vyhodnocení, jsou předány jako seznam názvů snímku, spolu jedinečným identifikátorem. Seznam je uložen jako JSON soubor pro snadnější předání do Python kódu.

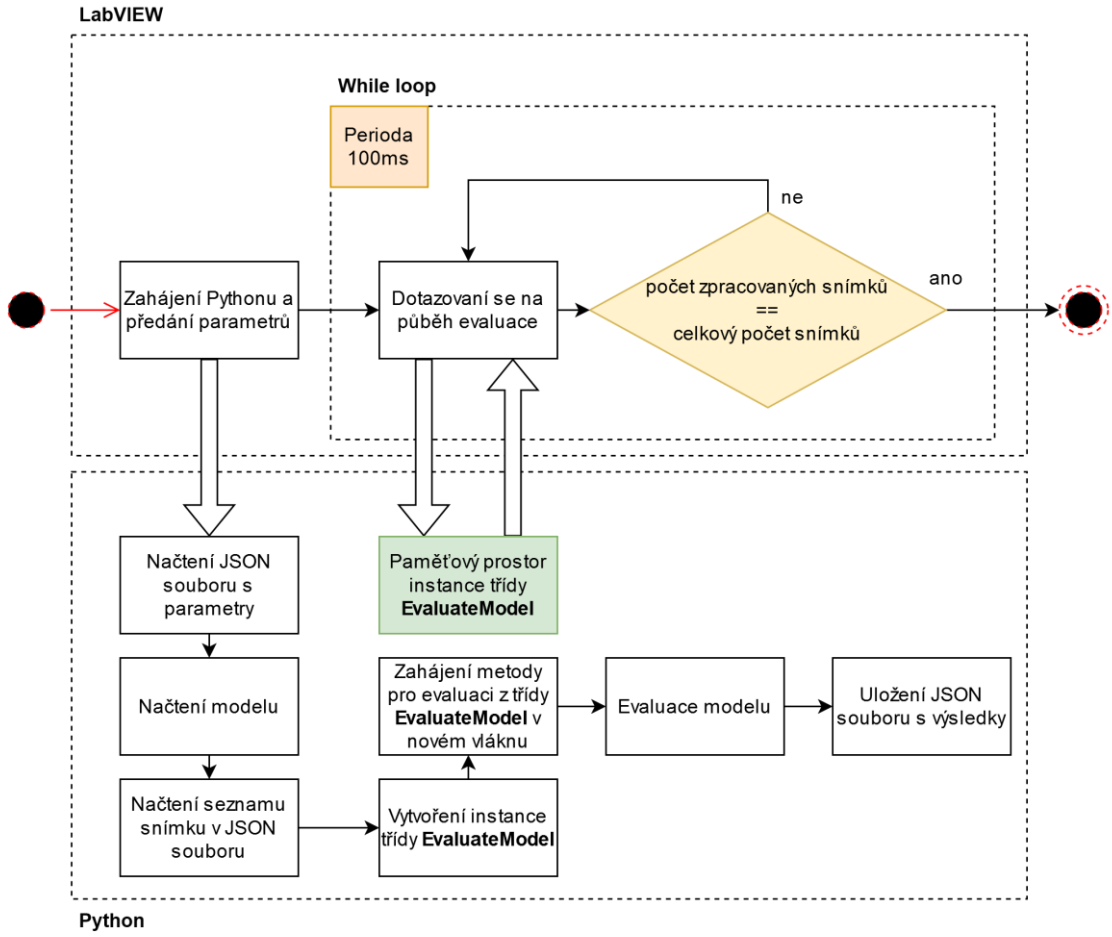


Obr. 4.25 *Aplikace pro evaluaci modelu*

Hlavní části aplikace:

1. Seznam modelů dostupných v projektu
2. Informace vybraného modelu ze seznamu modelů
3. Průběh evaluace
4. Průměrný čas za vyhodnocení jednoho snímku
5. Volba snímků, které se mají vyhodnotit – všechny v dataset, nebo jen testovací sada

Na Obr. 4.26 je ukázán diagram průběhu aplikace pro evaluaci modelu od zahájení tlačítkem *Start*. Po zahájení se provede nejdříve přesun veškerých parametrů do Python kódu, např. cesta ke složce se snímky apod. Postup přenosu do Python kódu je popsaný v podkapitole 3.5.2. V samotném Python metodě se nejdříve provede načtení JSON souboru s parametry k modelu a provedou se nutná nastavení. Následně se načte model. Po načtení model je možné načíst JSON soubor ve kterém je uložený seznam s názvy snímku určených pro evaluaci. Po načtení je možné vytvoření instance třídy EvaluateModel, které se předají veškeré nutné parametry jako je seznam snímků, model apod. Třída EvaluateModel byla vytvořena pro evaluaci dodaného model. Pro evaluaci je implementovaná metoda *evaluate*. Metodu evaluate je nutné spustit v samostatném vláknu, aby bylo možné provádět periodické čtení stavu evaluace. Na straně LabVIEW aplikace se po předání parametrů Python kódu, spustí WHILE smyčka, ve které se periodicky provádí dotazování průběhu evaluace (počet vyhodnocených snímků a průměrný čas). Vyčtená data se zapisují do ukazatele průběhu (bod 3 Obr. 4.25). Jakmile se rovná počet zpracovaných snímků s celkovým počtem snímků, tak se zastaví smyčka (na straně LabVIEW). Ještě potřeba se zmínit o metodě evaluate, kde při vyhodnocení snímku se výsledek z model ve formě predikční mapy se upraví (ořeže se na původní velikost, upraví se hodnoty pixelů) a výsledné mapy se uloží jako PNG obrázek do složky *prediction* (Obr. 4.16). Po dokončení evaluační smyčky na straně Pythonu se ještě provede uložení pole s výsledky klasifikačního skóre, kde každému je přiřazený odpovídající identifikátor pro spárování se snímky z databáze v LabVIEW aplikaci.



Obr. 4.26 *Diagram průběhu evaluace modelu*

4.6.3 Vyhodnocení predikcí

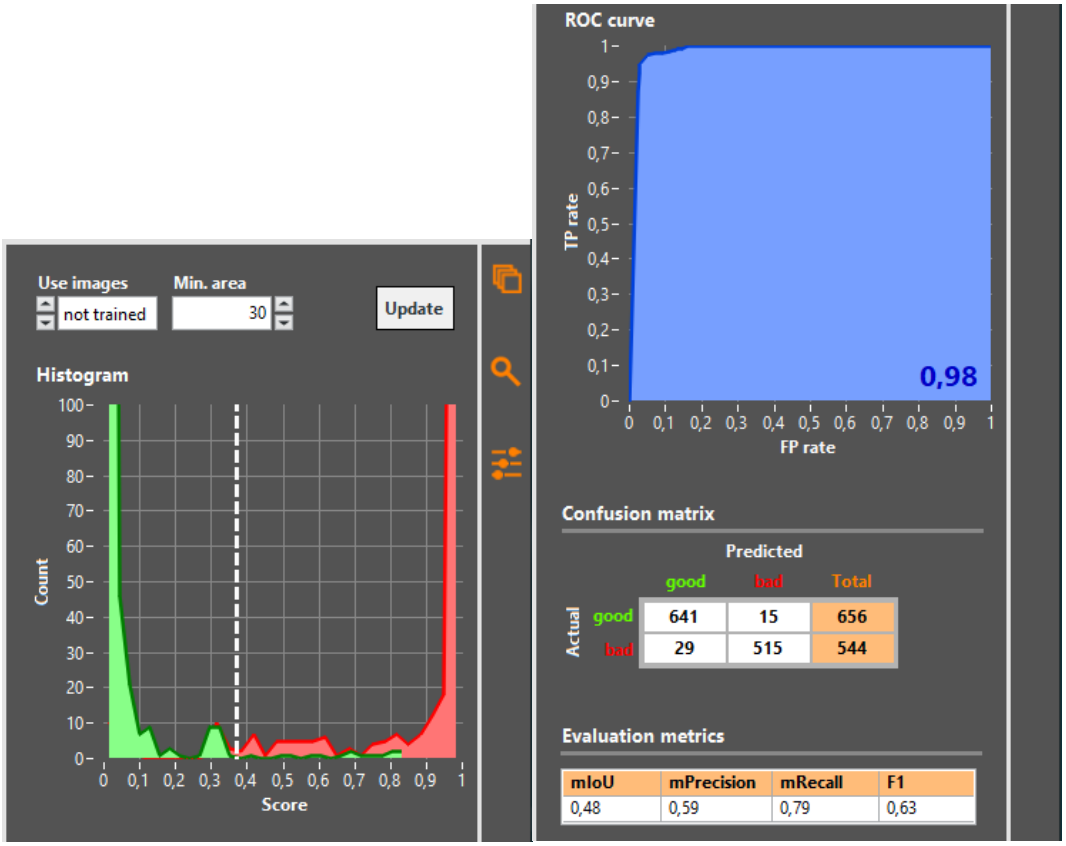
Po dokončení evaluace zvoleného modelu, která je popsaná v podkapitole 4.6.2, tak je možné vypočítat evaluační metriky v hlavní aplikaci. Metriky se nacházejí v okně (bod 6 Obr. 4.15) a je možné se k nim dostat přes lištu oken (bod 7 Obr. 4.15). Ukázka okna s evaluačními metrikami je na Obr. 4.27, je nutné poznamenat, že pro ukázku v diplomové práci bylo okno rozdělené na dvě části.

V horní části okna (Obr. 4.27) se nachází volba snímků, pro které se má provést výpočet. Výběr je *all labeled* (všechny anotované), *trained* (trénované), *not trained* (netrénované). V rámci predikčních map se může nacházet mnoho drobných objektů, které můžou být nežádoucí při výpočtu metrik. Z těchto důvodů je přidaný filtr predikovaných objektů, které odstraní všechny objekty menší než zadaná plocha v pixelech. Nakonec se nachází tlačítko Update, které provede výpočet evaluačních metrik.

Na levé části Obr. 4.27 se nachází histogram, který zobrazuje distribuci klasifikačních skóre. V histogramu jsou vykresleny dvě nezávislé křivky, první je křivka pro OK snímky a jejich predikční skóre (zelená křivka) a druhá křivka je pro NOK snímky a jejich predikční skóre (červená křivka). Součástí histogramu je vertikální posuvník, kterým se nastavuje prahová hodnota (threshold). Na základě prahové hodnoty se určuje hranice predikovaného klasifikačního skóre.

Na pravé části Obr. 4.27 už jsou evaluační metriky, které jsou teoreticky popsané v podkapitole 4.2. Horní graf je ROC křivka a v rohu křivky je hodnota AUC. Dále se nachází matice záměn, která je

silně závislá na nastavené prahové hodnotě. Poslední částí je tabulka s evaluačními metrikami pro segmentaci.

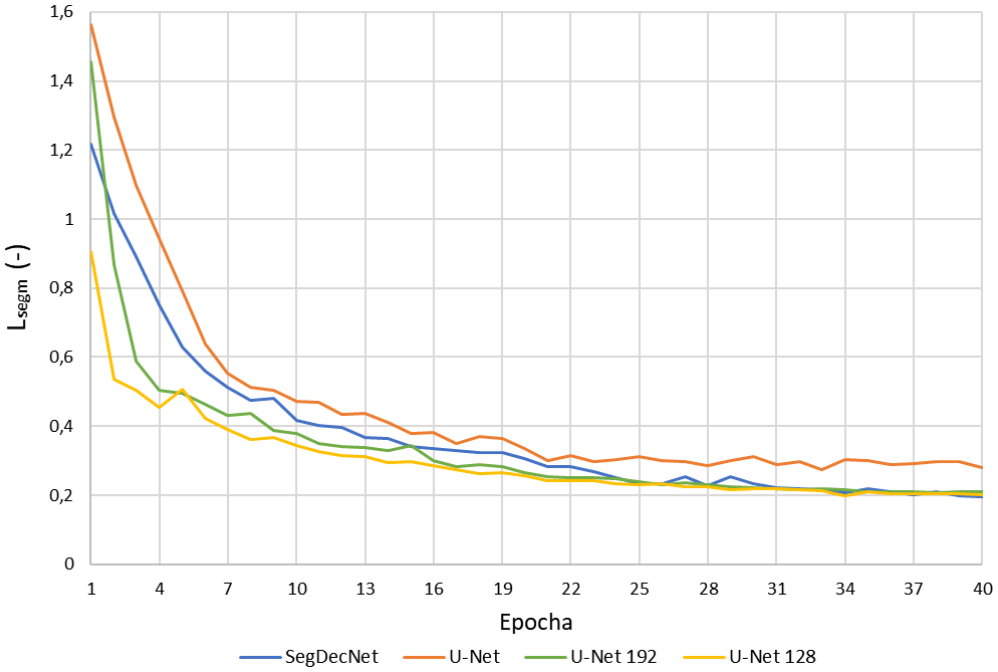


Obr. 4.27 *Okno pro evaluační metriky (kvůli rozměrům je rozdělené)*

5 Srovnání výsledků

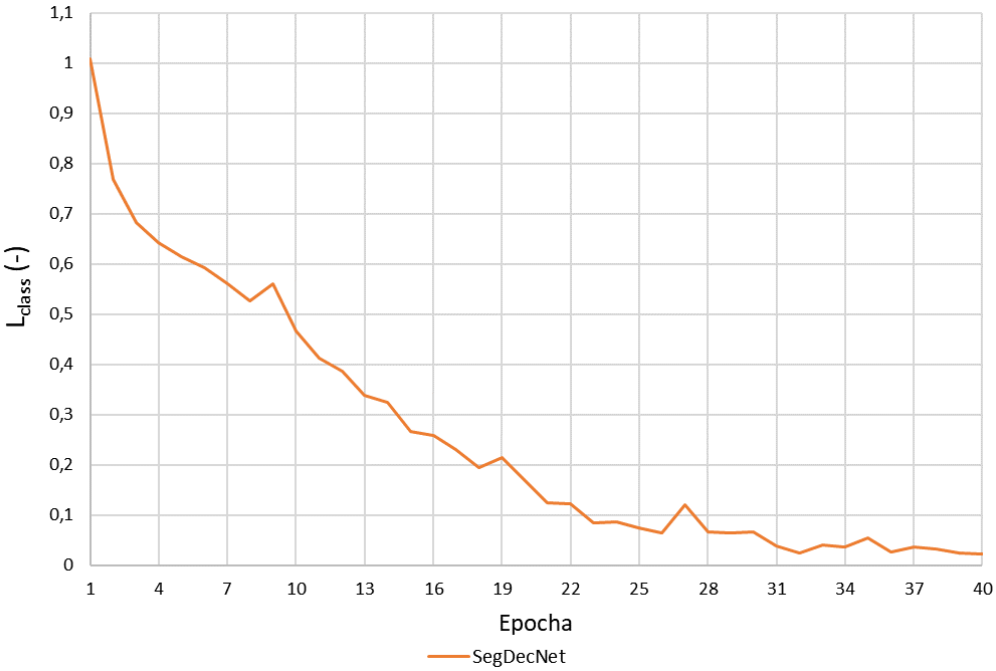
V této kapitole jsou rozvedeny výsledky natrénovaného modelu v Cognex DLS (4.3.1), a natrénované modely v jazyce Python s pomocí knihovny TensorFlow. Podrobný postup trénování je popsaný v podkapitolách 4.5.1 až 4.5.3.

Na Obr. 5.1 je ukázaný průběh ztrátové funkce pro segmentaci L_{segm}. V grafu jsou vykresleny pouze průběhy z trénovaní vlastních modelů. Trénování modelu v Cognex DLS neposkytuje údaje o ztrátové funkci. Pro všechny vlastní trénované modely je použita stejná ztrátová funkce kombinující binární křížovou entropii a Tversky s váhami $\alpha = 0{,}3$ a $\beta = 0{,}7$. Cílem je minimalizovat hodnotu ztrátové funkce. Z Obr. 5.1 vychází srovnatelně model SegDecNet, U-Net 192 a U-Net 128. Nejhorší výsledky při srovnání finálních hodnot ztrátových funkcí je v případě modelu U-Net využívající celé snímky. Naopak model U-Net založený na metodě patch-based s velikostí 128x128 ze všech modelů nejrychleji konvergoval.



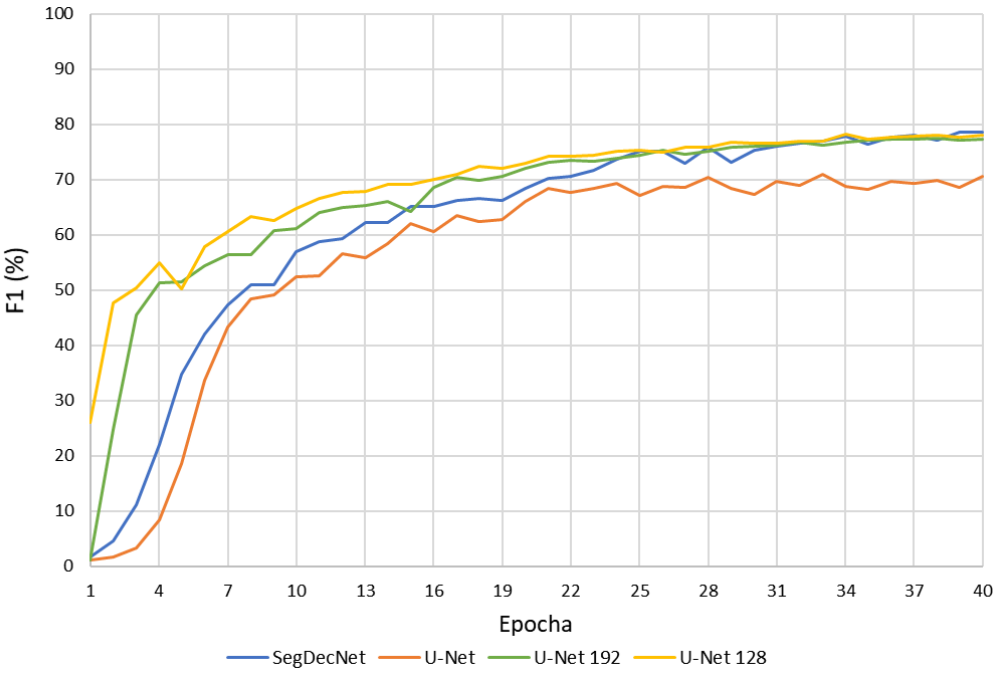
Obr. 5.1 *Průběh ztrátové funkce pro segmentaci během trénování modelů*

Na Obr. 5.2 je ukázka vývoje ztrátové funkce pro SegDecNet z klasifikační části modelu. Jako ztrátová funkce je použita pouze binární křížová entropie. Architektura U-Net neposkytuje klasifikační skóre, a tudíž nebyla u ní počítaná ztrátová funkce L_{class}.



Obr. 5.2 *Průběh ztrátové funkce pro klasifikaci během trénování SegDecNet*

Na Obr. 5.3 je ukázaná evaluační metrika F1, která byla vyhodnocena po každé zpracované epoše. Výsledky odrážejí průběh ztrátové funkce L_{segm} z Obr. 5.1. Nejlepší výsledky představovaly modely SegDecNet, U-Net 192 a U-Net 128 s dosaženou metrikou F1 téměř 80%. Nejhorší výsledky jsou u modelu U-Net využívající celé snímky, který dosáhl metriky F1 kolem 70%.



Obr. 5.3 *Průběh metriky F1 během trénování modelů*

V Tab. 5.1 jsou výsledky z metriky matice záměn pro všech pět natrénovaných modelů. Jedná se o výsledky, které byly získané evaluací modelů pouze na testovací sadě snímků. Žlutá barva v tabulce zvýrazňuje nejlepší výsledky za daný parametr. Hodnota TP představuje celkový počet správně

detekovaných snímků označených jako OK snímky. V ideálním případě by se hodnota TP rovnala celkovému počtu snímků označených jako OK snímky, tedy hodnotě 656. Nejblíže k této hodnotě je model SegDecNet s počtem 647. Druhá hodnota TN představuje celkový počet správně detekovaných snímků označených jako NOK snímky. V ideálním případě by se hodnota TN rovnala celkovému počtu snímků označených jako NOK, tedy hodnotě 544. Nejblíže je k této hodnotě byl model z Cognex DLS s počtem 534.

Vzhledem ke snaze detekovat defektní SMD součástky je důležité, aby byly odhaleny všechny snímky, který byly označeny jako NOK kusy. Proto musí být hodnota FP ideálně nulová, protože představuje počet snímků označených jako NOK, ale predikovaných jako OK snímky. Nejblíže je k této ideální hodnotě model z Cognex DLS.

Hodnota FN je v ideálním případě nulová. Přestavuje situaci, kdy snímek je OK, ale je predikovaný jako NOK. Nejlepší výsledky dosahoval model SegDecNet.

Model z Cognex DLS poskytuje na výstupu klasifikační skóre, které bylo využité pro rozřazení do kategorií TP, TN, FP a FN. Prahová hodnota T byla zvolena experimentálně pro dosažení optimálních hodnot. Druhý model SegDecNet, také poskytuje na výstupu klasifikační skóre, ale které představuje pouze dílčí hodnotu finálního klasifikačního skóre. Finální skóre je počítané kromě výstupního skóre z model, i na základě nalezených objektů v predikční mapě. Přičemž nalezené objekty musí mít plochu větší než zvolená minimální plocha 80 px^2. Hodnota minimální plochy byla experimentálně nalezena pro dosažení optimálních výsledků.

Modely s architekturou U-Net neposkytují na výstupu klasifikační skóre, ale pouze predikční mapu. Na základě predikční mapy bylo zhodnocené, jestli se jedná o OK, nebo NOK snímek. V predikční mapě se provedlo hledání objektů větších, než je zvolená minimální plocha. Pokud je nalezený alespoň jeden objekt, tak se vyhodnotil jako NOK snímek.

Metrika hodnotící kvalitu binárního klasifikátoru ROC-AUC ukazují nejlepší výsledky pro model z Cognex DLS a v závěsu s modelem SegDecNet. Je dobré připomenout, že metrika ROC-AUC je invariantní na zvolené prahové hodnotě T (viz 4.2.2), oproti tomu matice záměn je silně závislá na prahové hodnotě T.

Tab. 5.1 *Výsledné evaluační metriky pro klasifikační úlohu*

Model	TP	TN	FP	FN	Min. plocha (px^2)	T (-)	ROC-AUC (%)
U-Net 128	622	517	27	34	120	-	95
U-Net 192	624	520	24	32	80	-	95
U-Net	623	515	29	33	80	-	95
SegDecNet	647	519	25	9	80	0,4	98
Cognex DLS	638	534	9	22	-	0,48	99

V Tab. 5.2 je ukázka evaluačních metrik pro segmentaci. Model U-Net a jeho varianty dosahovaly v klasifikaci horších výsledků. Naproti tomu v segmentaci dosahují hlavně modely U-Net s metodou patch-based nejlepších výsledků v metrikách IoU, F1 a Recall. Horší výsledky, ale dosahují v průměrných časech $\mathbf{t_{process}}$ za vyhodnocení jednoho snímků. Důvod je nutnost vyhodnotit větší počet

snímků. Například u modelu U-Net 128 je použitá velikost okna 128x128 a překryv 80 pixelů. Výsledný počet vyextrahovaných patches je potom 36.

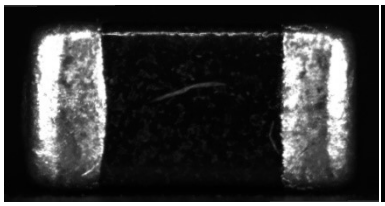
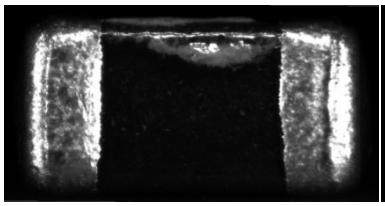
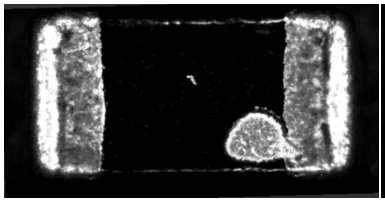
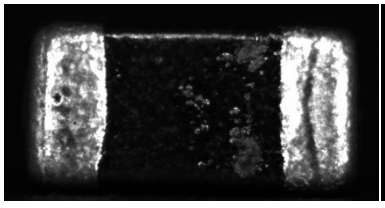
Model SegDecNet poskytuje ve srovnání s ostatními modely nejvíce vyváženými výsledky z metrik a průměrného času za snímek.

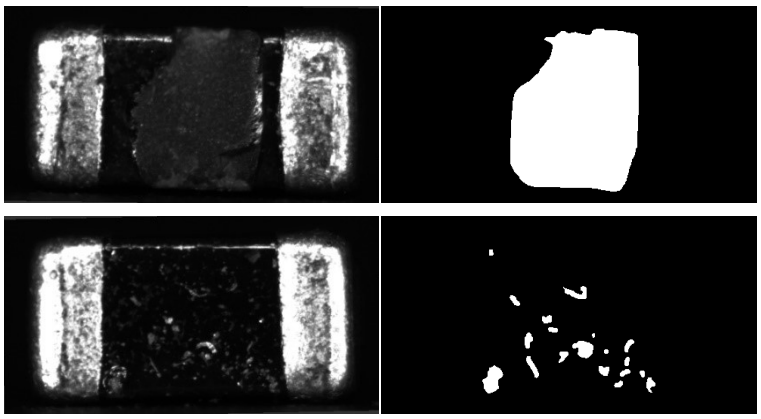
Tab. 5.2 *Výsledné evaluační metriky pro segmentační úlohu*

Model	IoU (%)	F1 (%)	Recall (%)	Precision (%)	$t_{process}$ (ms)
U-Net 128	49	64	83	57	78
U-Net 192	48	62	81	56	32
U-Net	46	61	79	57	12
SegDecNet	49	63	80	59	20
Cognex DLS	47	59	60	77	16

V Tab. 5.3 je ukázka snímků z testovacího datasetu SMD součástek. Ke každému snímku je i ukázaná příslušná anotace v podobě binární segmentační mapy.

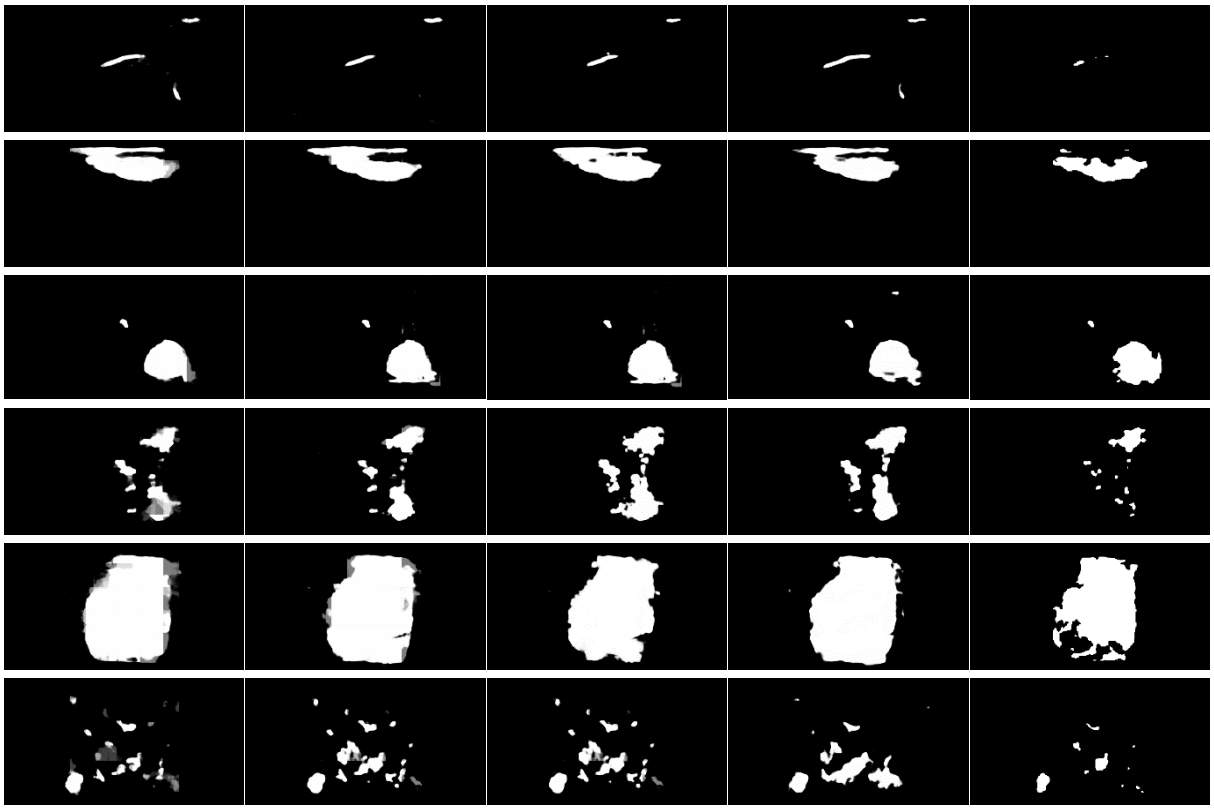
Tab. 5.3 *Ukázka snímků a anotací*

Snímek	Anotace
	
	
	
	



V Tab. 5.4 jsou ukázané predikční mapy z jednotlivých modelů na snímcích z Tab. 5.3.

Tab. 5.4 *Predikční mapy na snímcích z Tab. 5.3*

U-Net 128	U-Net 192	U-Net	SegDecNet	Cognex DLS
				

Závěr

Cílem diplomové práce je rozbor problematiky strojového učení pro vizuální inspekce výrobků s cílem detekce defektů. Dosažení cíle bylo realizované v několika krocích. Na začátku se provedla rešerše publikací, které se zabývají primárně technikami hlubokého učení pro vyhodnocení defektů na výrobcích. Následně se provedl obecný teoretický rozbor oblasti hlubokého učení se zaměřením na konvoluční sítě. Dále byl provedený rozbor využitých nástrojů pro natrénování model. Práce se zaměřila na porovnání celkem dvou přístupů. První byl ve využití komerčního programu Cognex DLS a druhý v implementaci open source architektur pomocí knihovny TensorFlow. Implementovány byly architektury U-Net a SegDecNet, který byly částečně upraveny pro potřeby zvoleného datasetu snímků. Zároveň v případě architektury U-Net byly otestovány dvě metody úpravy snímků. První byla ve využití celých snímků a druhá metoda spočívala v rozdělení snímku na menší části (metoda patch-based). Po natrénovaných model byla vytvořen aplikace pro testování modelů, a to včetně výpočtu evaluačních metrik pro zhodnocení kvality jednotlivých modelů na testovacím datasetu.

Pokud bychom hodnotili modely podle přesnosti klasifikace, tak nejlepších výsledků dosáhl model z Cognex DLS. Jenom o trochu horší výsledky byly u modelu SegDecNet a to hlavně u počtu FP. V případě U-Net model s metodu celých snímků a patch-based nejsou žádné signifikantní rozdíly. Všechny varianty U-Net jsou výrazně horší než modely SegDecNet a z Cognex DLS.

Pokud bychom hodnotili modely podle přesnosti segmentace, tak nejlepší výsledky byly z modelu U-Net s metodou patch-based s oknem 128x128. Ostatní modely kromě modelu z Cognex DLS, jsou pouze maximálně o 2% horší ve zvolených metrikách. Metoda patch-based rozhodně zlepšila rychlost konvergování modelu a lehce zvedl výsledky evaluačních metrik. Model z Cognex DLS byl ve všech metrikách pro segmentaci horší , až na metriku Precision, ve které dosahoval nejvyšší hodnoty ze všech modelů.

Výsledky ukazují, že je možné i s open source architekturami dosáhnout v případě klasifikace téměř srovnatelných výsledků jako z komerčního programu Cognex DLS. V případě čisté segmentace je možné i překonat výsledky z Cognex DLS.

Jedním z možných rozšíření může být využití tzv. maskování obrazu. Nejdříve by se provedlo maskování oblasti terminací a následně by se natrénoval model pouze na vyhodnocení defektů na keramickém materiálu. Potom by se provedlo maskování keramického materiálu a provedlo by se trénování dalšího modelu pro vyhodnocení defektů pouze na terminacích. Rozdělením úlohy na dva modely by mohlo vést ke zlepší výsledků, protože by se každý model specializoval na defekty na terminaci, anebo keramickém materiálu.

Další rozšíření může být využití tzv. GAN (Generative Adversarial Networks). Tato architektura je po natrénovaní schopná generovat syntetické snímky. Architektura GAN by se mohl natrénovat na defektních snímcích. Vygenerované syntetické snímky je potom možné použít pro rozšíření trénovacího datasetu využitého pro modely s cílem detekce defektů a mohlo by se tím reálně vylepšit výsledky natrénovaných modelů.

Literatura

[1] VLACH, Jaroslav, Josef HAVLÍČEK a Martin VLACH. *Začínáme s LabVIEW*. Praha: BEN - technická literatura, 2008. ISBN 978-80-7300-245-9.

[2] BUSLAEV, Alexander, Vladimir I. IGLOVIKOV, Eugene KHVEDCHENYA, Alex PARINOV, Mikhail DRUZHININ a Alexandr A. KALININ. Albumentations: Fast and Flexible Image Augmentations. *Information* [online]. 2020, **11**(2) ISSN 2078-2489. Dostupné z: doi:10.3390/info11020125

[3] *TensorFlow* [online]. Dostupné také z: https://www.tensorflow.org/

[4] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[5] VisionPro Deep Learning Help [online]. Cognex Corporation, 2020, Dostupné z: https://support.cognex.com/docs/deep-learning_101/web/EN/deep-learning/Content/deep-learning-Topics/get-started/get-started.htm

[6] RONNEBERGER, Olaf, Philipp FISCHER a Thomas BROX. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* [online]. Cham: Springer International Publishing, 2015, 2015-11-18, , 234-241 [cit. 2021-04-15]. Lecture Notes in Computer Science. ISBN 978-3-319-24573-7. Dostupné z: doi:10.1007/978-3-319-24574-4_28

[7] TABERNIK, Domen, Samo ŠELA, Jure SKVARČ a Danijel SKOČAJ. Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing* [online]. 2020, **31**(3), 759-776 ISSN 0956-5515. Dostupné z: doi:10.1007/s10845-019-01476-x

[8] JADON, Shruti. A survey of loss functions for semantic segmentation. *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* [online]. IEEE, 2020, 27. 10. 2020, , 1-7. ISBN 978-1-7281-9468-4. Dostupné z: doi:10.1109/CIBCB48159.2020.9277638

[9] *Hands-On Convolutional Neural Networks with Tensorflow*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78913-033-1

[10] GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems* [online]. Beijing: O'Reilly, [2017]. ISBN 978-1-491-96229-9.

[11] WEIMER, Daniel, Bernd SCHOLZ-REITER a Moshe SHPITALNI. Design of deep convolutional neural network architectures for automated feature extraction

in industrial inspection. *CIRP Annals* [online]. 2016, **65**(1), 417-420. ISSN 00078506. Dostupné z: doi:10.1016/j.cirp.2016.04.072

[12] LIU, Yu-ting, Ya-ning YANG, Chao WANG, Xiang-yu XU a Tao ZHANG. Research on Surface Defect Detection Based on Semantic Segmentation. *DEStech Transactions on Computer Science and Engineering* [online]. 2019, (aicae). ISSN 2475-8841. Dostupné z: doi:10.12783/dtcse/aicae2019/31504

[13] LIN, Hui, Bin LI, Xinggang WANG, Yufeng SHU a Shuanglong NIU. Automated defect inspection of LED chip using deep convolutional neural network. *Journal of Intelligent Manufacturing* [online]. 2019, **30**(6), 2525-2534. ISSN 0956-5515. Dostupné z: doi:10.1007/s10845-018-1415-x

[14] QU, Zhenshen, Jianxiong SHEN, Ruikun LI, Junyu LIU a Qiuyu GUAN. PartsNet. *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence - CSAI '18* [online]. New York, New York, USA: ACM Press, 2018, 2018, , 594-599. ISBN 9781450366069. Dostupné z: doi:10.1145/3297156.3297190

[15] LIU, Yang, Ke XU a Jinwu XU. Periodic Surface Defect Detection in Steel Plates Based on Deep Learning. *Applied Sciences* [online]. 2019, **9**(15). ISSN 2076-3417. Dostupné z: doi:10.3390/app9153127

[16] TAO, Xian, Dapeng ZHANG, Wenzhi MA, Xilong LIU a De XU. Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks. *Applied Sciences* [online]. 2018, **8**(9). ISSN 2076-3417. Dostupné z: doi:10.3390/app8091575

[17] *LabVIEW 2018 Help* [online]. NATIONAL INSTRUMENTS. Dostupné také z: https://zone.ni.com/reference/en-XX/help/371361R-01/

[18] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep Learning* [online]. MIT Press, 2016 [cit. 2021-4-30]. Dostupné z: https://www.deeplearningbook.org/

[19] LIN, Min, Qiang CHEN a Shuicheng YAN. *Network in network* [online]. 2013. Dostupné také z: https://arxiv.org/abs/1312.4400

[20] Convolutional Neural Networks (CNNs / ConvNets). *CS231n: Convolutional Neural Networks for Visual Recognition* [online]. Stanford CS. Dostupné z: https://cs231n.github.io/convolutional-networks/

Přílohy

Veškeré přílohy jsou připojeny v elektronické podobě.

Adresář elektronické přílohy obsahuje podadresáře:

- *project_evaluation*: jedná se LabVIEW projekt, který obsahuje vytvořenou aplikaci pro testovaní natrénovaných modelů
- *project_training*: jedná se o Python projekt, který obsahuje soubory použité pro natrénovaní použitých neuronových sítí
- *export_prediction*: jedná se LabVIEW projekt, který obsahuje aplikaci pro vyhodnocení modelu z Cognex DLS a zároveň veškerých predikcí